NASA Contractor Report 178363

Development of N-Version Software Samples for an Experiment in Software Fault Tolerance

L. Lauterbach

Software Research and Development
Center for Digital Systems Research
Research Triangle Institute
Research Triangle Park, North Carolina 27709

Contract NAS1-17964
Task Assignment No. 4
September 1987

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

(NASA-CR-178363) DEVELOPMENT OF N-VERSION SOFTWARE SAMPLES FOR AN EXPERIMENT IN SOFTWARE FAULT TOLERANCE Final Report (Research Triangle Inst.) 179 p Avail: NTIS HC A09/MF A01 CSCL 09B G3/61

N87-30096

Unclas
0102768

NASA Contractor Report 178363

Development of N-Version Software Samples for an Experiment in Software Fault Tolerance

L. Lauterbach

Software Research and Development
Center for Digital Systems Research
Research Triangle Institute
Research Triangle Park, North Carolina 27709

Contract NAS1-17964
Task Assignment No. 4

September 1987

Table of Contents

List of Figures ii
List of Tables ii

1. Introduction 1
1.1. Background 1
1.2. Research Goals 2
1.3. Task Summary 2
1.4. Purpose of This Document 3
1.5. Participants 3

2. Project Design 4
2.1. Evolution of Project Design 4
2.2. The Application 4
2.2.1. A Description of the Problem 5
Geometry 6
Calculations and Outputs 8
2.3. Specifications 8
2.4. The Student Subjects 9
2.5. The Development Environment 9
2.5.1. Introduction 9
2.5.2. On-line Environment 9
2.5.3. General Environment 11
2.5.4. Communications Protocol 11

3. Task Conduct 12
3.1. Time Schedule 12
3.2. Design Phase 13
3.3. Code and Unit Test Phase 14
3.4. Integration Test Phase 14
3.5. Acceptance Test Phase 14
3.5.1. Input Distribution 14
3.5.2. Input Data 15
3.5.3. Acceptance Test Harness 16
3.5.4. Acceptance Test Protocol 16

4. Data 17
4.1. Descriptive Data 17
4.1.1. Subjects 17
4.1.2. Specification Question-and-Answer Sessions 17
4.1.3. Versions 19
4.2. Empirical Data 20

5. Data Analysis 20
5.1. Preliminary Analyses 20
5.1.1. Final Acceleration Estimates 20
5.1.2. System States 21
5.2. Discussion 22

6. Conclusions 22
6.1. Other Data Analyses 22
6.1.1. Error Analysis 23
6.1.2. Fault Analysis 23
6.2. Other Studies 23
6.3. Task Design Improvements 24
6.3.1. Reduce Geographic Separation of Sites 24
6.3.2. Complete Experimental Tools Early 25
6.3.3. Conduct Trial Run 25
6.3.4. Improve Software Development Environment 26
6.4. Summary 26

References 23

Appendix A. RSDIMU Specifications 29
Appendix B. Student Question and Answer Sessions 94

List of Figures

Figure 2-1. Relation of I-Frame to Semi-Octahedron 6
Figure 2-2. Relation of A-Frame to A-Face of Semi-Octahedron 7
Figure 3-1. Program Development Schedule 12

List of Tables

Table 2-1. Programmer Profile 10
Table 4-1. Partitioning of Student Questions 18
Table 4-2. RSDIMU Version Structure 19
Table 5-1. SYSSTATUS Results on 25,000 Test Cases 22

1. Introduction

1.1. Background

Advanced avionics systems are relying on software to an increasing degree for critical flight functions as analog implementations are being replaced by digital implementations of functions and both military and commercial aircraft are moving to fly-by-wire systems. Fault tolerant methods have been studied as means for improving the reliability of these and other complex, critical software systems through use of redundant software versions.[1] The recovery block and N-version methods of incorporating fault tolerance into the system acknowledge that faults are likely to remain in the production software, and although these faults cannot be avoided, their resultant errors can be tolerated.

An original view of software fault tolerance that promised great improvements in reliability was that software systems could be modeled like redundant hardware systems. That is, through the use of reliable, redundant versions in an N-version framework, it was hypothesized that independently designed software versions would exhibit statistically independent failures. While independence does not imply mutual exclusion (versions that fail independently *may* fail at the same time with a given probability), it does mean that imperfect versions with a reliability of over 0.5, incorporated into an N-version system, will result in a system reliability gain over an individual version. Thus, statistically independent failures among N fairly reliable versions in a fault tolerant framework (along with a perfect voter) would result in a lower system failure rate than one of the N versions running alone. In addition, such a fault-tolerant system would result in a lower failure rate than redundant versions running in an N-version framework where correlated failures are seen.

Study of fault-tolerant software research shows that a recent experiment involving students on the east and west coasts of the United States has supported the rejection of the hypothesis that independently-developed software versions exhibit independent errors.[2,3] Eckhardt and Lee[4] have recently developed a mathematical model that accounts for *coincident* errors (failures in multiple versions executing on the same input) among independently designed software versions without assuming statistically independent failures by the versions. They have provided hypothetical examples to show that the number of versions to use in a multiversion system for highest reliability is a function of the intensity of coincident errors. They define *coincident* errors as those resulting from either: a) multiple faults which produce dissimilar outputs but are manifested by the same input conditions, or b) related software design faults causing identical incorrect outputs.[4] Thus, the term accounts not only for the chance simultaneous errors as may occur among independently failing versions, but also for related specification or design faults that result in correlated errors. The extent to which coincident errors occur in different N-version systems is unknown; data from several different systems is needed to begin to study this factor.

1.2. Research Goals

This effort was initiated to obtain multiple, independently-developed versions of an avionics application; to gather failure data on these versions; to plan for the use of these data in analyzing the failures of the versions; and to identify additional research objectives for future work. This task was conducted as a part of the Critical Activating Technology (C-A-T) Program of research.

1.3. Task Summary

Twenty versions of an inertial measurement unit problem coded from a single English language specification were obtained, as well as design documents, design and code walk-throughs, and the other written deliverables (see Table 3-1) through this task.

Experimental subjects submitted over 202 questions concerning the specifications; 36% of these revealed specification ambiguities, and 12% revealed errors. Because of the volume of questions received and the fact that they were from several different teams at each university, it is likely the N-version environment contributed to a debugging of the specifications. However, we cannot assume that the specifications are now error-free.

Preliminary data analysis reveals occurrences of coincident errors (see Section 5.1.2.). At least a portion of these are *not* attributable to specification errors, but instead to non-adherence to the specifications. However, neither detailed data analysis nor fault analysis were called for under this task, and future work may reveal the specifications as a source of errors. Kelly and Avizienis report that their preliminary data analysis has uncovered remaining specifications errors.[5]

To further analyze real number outputs of the versions, a methodology for determining correctness must be devised. A tolerance, epsilon, in the final floating point output comparisons is necessary but not sufficient; small deviations in intermediate floating point outputs may result in discrepancy among intermediate binary decisions. These intermediate binary decisions may greatly affect later computations of final outputs. A method for voting on intermediate results must also be devised. In future specifications for use in N-version programming, specification of the precision and accuracy required of output variables would obviate this problem.

Also of note is the importance of developing and testing software tools well in advance of the time they are needed. Tool development was rushed because of hard time constraints imposed by employing students during their vacation months; as a result, the acceptance test was not as complete as was desired, and the intended gold version had not been subjected to the amount of testing necessary to certify it as a golden version.

These versions provide raw data that can be used in comparing reliability models, determining single and N-version system reliabilities, and performing error and fault analyses. The written documents obtained from the subjects as a part of this task can

be scrutinized to compare design methodologies, to compare overall methodologies and reliabilities achieved, and so forth. Several potential future studies are discussed in the body of this report.

1.4. Purpose of This Document

This report contains a thorough description of the fault tolerant software development and presents the data gathering and analysis done as a part of this task.

Equal in importance to the data gathered and their analyses are the documentation of design strategy for obtaining the software versions, the problems encountered during design and execution of this task and the workability of their solutions. Experimentation involving software sample development is a relatively new endeavor, and there is much to be learned to improve design, controls, conduct, protocol, and quality of data gathered.[3] We hope the information provided herein will allow others to learn from our experience, as well as from our data and results.

1.5. Participants

The design and conduct of this effort to gather N-version software samples was largely a group effort. The jobs and the people who performed them are listed below. Principal investigators involved were: Dr. Roy Campbell at the University of Illinois at Urbana-Champaign (UIUC), Dr. Dave McAllister at North Carolina State University (NCSU), Dr. John Kelly at the University of California at Los Angeles (UCLA), Dr. John Knight at the University of Virginia (UVA), and co-principal investigators Dr. John McHugh and Linda Lauterbach at Research Triangle Institute (RTI).

Dr. Dave Eckhardt was the NASA-LaRC contract monitor. Dr. Larry Lee, also of NASA-LaRC, contributed to the planning phase. Dr. Alper Caglayan and Greg Zacharias of Charles River Analytics, Inc. (CRA) provided consulting services during the task design and software development stages.

At RTI's Center for Digital Systems Research (CDSR) the following people also contributed to this task: Janet Dunham, Senior Research Computer Scientist, and John Pierce, Manager of the Department of Software Research and Development, provided consulting advice; and Jeff Bartlett, Research Engineer, and Don Rich, Programmer Analyst, developed software tools.

Experiment aides helped with both technical and administrative work at each university. They were: Hal Render at UIUC, Dr. Mladen Vouk at NCSU, Rung-Tsong Lyu at UCLA, and Beth Stubbs at UVA.

Forty graduate students developed the versions of the RSDIMU problems used in this study. They were: M. Barr, D. Bellows, D. Carney, N. Covey, M. Devarakonda, A. Dollas, J. Grass, J. Graver, A. Liu, and M. Schmitz from UIUC; K Boone, M. Chao, Y. Choe, M. Davis, R. Harwell, Jr., V. Janakiram, S. S. Ku, S. Oliver, A. Paradkar, and J. Yih-Liang from NCSU; M. Aghdassi, M. Chen, C. Ip, T. Lee, C. Lin, K. Rong, B. Swain, A. Tai, B. Ulery, and W. Yau from UCLA; and K. Bass, P. Dickenson, T.

Donnelly, G. Fisher, H. Morel, J. Peden, D. Reape, M. Rouleau, J. Thomas, and C. Zacharewicz from UVA.

2. Project Design

2.1. Evolution of Project Design

Under ideal conditions code development of this type would be obtained by employing experienced avionics systems engineers to design and code an actual avionics application under their normal working environments. Although this was the intended scope initially, the decision to conduct this task at multiple universities, employing graduate students as subjects, was made early in the requirements analysis. There were two main reasons for this. First, there were found to be very few avionics engineers available to employ as programmers, and their salaries were higher than could be paid under allotted funds. Second, the nature of funding at NASA provided money especially earmarked for grants to universities; it could be used to pay graduate students, but not to pay professional programmers in the avionics field.

The full design of this study was accomplished through several group meetings of the principal investigators. Early on, the following qualities of the study were determined:

- The application chosen must be representative of avionics software, yet must be appropriate for graduate students.
- An English language functional specification of the application will be provided.
- A controlled development environment and protocol must be set up and monitored since programmers will be working in different geographic areas.
- Computer Science graduate students will be employed at the University of California at Los Angeles (UCLA), the University of Virginia (UVA), North Carolina State University (NCSU), and the University of Illinois at Urbana-Champaign (UIUC) to develop the versions and will work in teams of two.

2.2. The Application

In addition to the major requirement that the application be from the avionics field, it should also be suitable for graduate students who have no avionics-specific knowledge. Most avionics software is written in a low-level language for specialized hardware; university students would likely not be familiar with either the language or the hardware. The study was not envisioned as including and/or accounting for learning effects from students beginning the design and coding with unfamiliar tools and environment, so the need of being able to code the problem in a high-level language commonly used at universities and on hardware commonly available at universities became apparent.

Many avionics programs perform time-critical real time functions, including closed-loop control tasks. These attributes were disallowed from the application

because of the constraint that incorporating fault tolerance into the operational system should not significantly alter the functioning of the system. Two undesirable possibilities thus avoided were 1) the case where adding fault tolerance to time-critical computations would slow the performance to unacceptable speeds, and 2) the case where adding fault tolerance to closed loops might alter the behavior of the loop to the point that the system could become unstable.

Since the students would be employed for either one semester or during the summer, a problem that required approximately five man-months of effort was needed. This estimate of effort was arrived at by assuming that programmers would work in teams of two; it would then take about ten weeks for a team to complete the design, code, integration test, and acceptance test phases of the task. We wanted the application to be easily increased or decreased in size and scope in case our initial judgment of effort was off. We also wanted one with considerable complexity, so the students' work would not be trivial and result in error-free programs.

It was necessary that we find an application for which complete, unambiguous specifications either existed or could be written in the allotted time. We wanted a problem with a large input space in order to preclude the students' exhaustively testing their programs, and we wanted a problem with a large output vector to allow voting on several computational results. We required that the problem have well-defined acceptance criteria to make it possible to test for these criteria before accepting a version as "finished." Also required was that the application have unique outputs or real number outputs within a determinable margin of error *epsilon*, so the versions could be used with an N-version voter and would be suitable for automated testing. It was noted that an application with both specifications and a well-tested "golden" version available would save a significant amount of time, as creation of these two items could take months; a specification was seen as mandatory for this task, and a golden version, as highly desirable.

The Redundant Strapped Down Inertial Measurement Unit (RSDIMU) problem was eventually chosen after a long search as meeting most of the requirements above. No specifications existed, so functional specifications were drafted and reviewed (see Appendix A for the complete functional specifications). Neither did a golden version exist. Time constraints did not allow for design and coding of a golden version before the students began work; however, the fact that the algorithm could be solved in reverse pointed to a potentially less complex and time-consuming way of determining version correctness than solving the forward solution.

2.2.1. A Description of the Problem

The complete specifications, as distributed to the subjects, are included as Appendix A. The following is a brief summary of the problem, meant to give the reader an immediate, if incomplete, understanding of the complexity of the problem and the types of computations involved.

Geometry

The iISDIMU can be visualized as a four-sided, regular pyramid. (See Figure 2-1 and Figure 2-2.) There are eight linear accelerometers on the pyramid; two on each triangular face. The sensors on any one face are located on the face centroid (it is assumed for simplicity that they occupy the same point in space). The sensors measure acceleration at 90° to each other along the face; each measures at a 45° angle from the perpendicular bisector of the base edge measuring in the direction of the base edge.

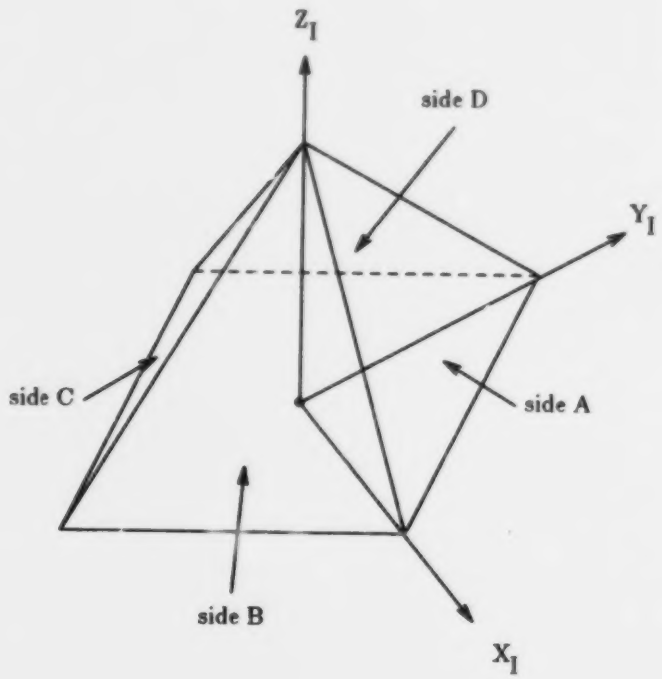


Figure 2-1. Relation of I-Frame to Semi-Octahedron

Several coordinate systems are defined and used in this problem: the navigation frame of reference, vehicle frame of reference, instrument frame of reference, idealized sensor frames of reference, and actual measurement frames of reference. All of these systems are orthogonal except the measurement frames, which are nonorthogonal by only a few degrees at most.

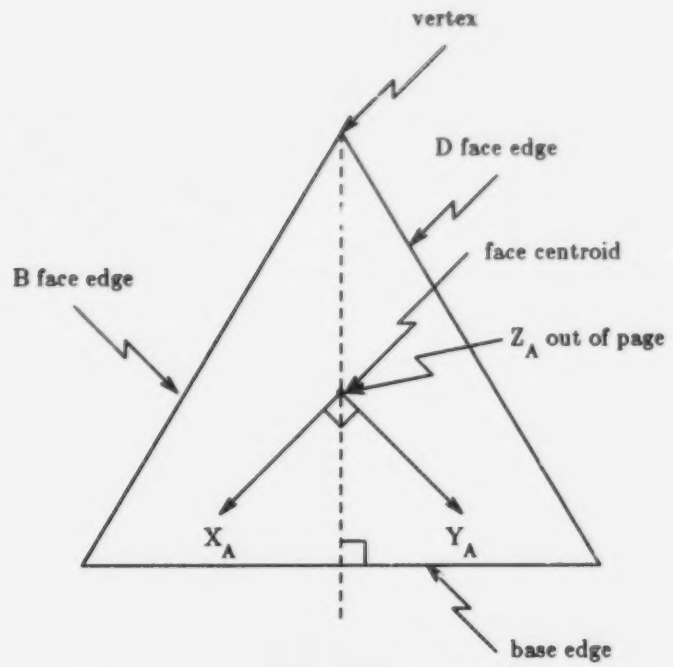


Figure 2-2. Relation of A-Frame to A-Face of Semi-Octahedron

Calculations and Outputs

In general, calculations include simple statistical procedures and linear algebra with emphasis on matrix operations. Calculations require attention to numerical considerations.

The sensors measure linear acceleration along the measurement frames of reference in nonnegative, integral units of "counts." Included in the input data set are sensor calibration data, an instantaneous in-flight sensor measurement for operable sensors, and information showing which sensors were working properly as of the last sampling. The procedure must initially analyze the sensor calibration data to see if any sensors must be marked as failed because they are out of calibration. This corresponds to the on-ground, preflight calibration performed in aircraft.

The subsequent in-flight sensor measurement is then compensated for misalignment of the sensors on the pyramid (transformed from measurement to idealized sensor frames of measurement), and this compensated, in-flight measurement of the working sensors is processed. First, sensors whose noise component exceeds a given standard deviation are marked as failed and are excluded from further computations. Next, an edge vector test is performed using adjacent faces with two working sensors to determine if any sensor's current in-flight acceleration measurement is faulty. Finally, least squares estimation with all good sensors is used to calculate acceleration. Acceleration is reported along the navigation frame's x, y, and z axes.

Various other outputs are calculated, such as acceleration estimates obtained by grouping combinations of sensors along defined channels; vectors showing how much noise each sensor measured, which sensors were failed at preflight calibration, and which were failed as a result of noise; and the operational status of the measurement system. Variables that drive a cockpit display are also outputs.

2.3. Specifications

The specifications were outlined and iteratively reviewed and refined in group meetings and by electronic mail, over a couple of months. The majority of the specification was written by two people; it consists of three chapters comprising 42 pages, and five appendices comprising 24 pages. The complete specification is included in Appendix A of this report.

The mathematical complexity of the specified problem, contrasted with the average mathematical background of the subjects, resulted in a decision to include outlines of some applicable mathematical methods in appendices to the specifications. While it was recognized that true specifications state only *what* the requirements are, the principal investigators saw a need to provide some high-level design information on *how* to meet the requirements for a portion of the problem.

Note that subsequent queries from subjects on the methodology in the specifications were answered by stating that the methods provided in the appendices were optional and explanatory only; other methods that could be used to meet the

specifications were allowable. This response was a compromise to allow for maximum design diversity while providing sufficient information for understanding the problem.

2.4. The Student Subjects

A total of 40 student subjects were hired, ten each at the University of California at Los Angeles (UCLA), University of Illinois at Urbana-Champaign (UIUC), University of Virginia (UVA), and North Carolina State University (NCSU). All were pursuing graduate degrees in computer science. They came from diverse backgrounds and had varying levels of course work in computer science and mathematics and had varying levels of work experience in the computer science field. Descriptive profiles of 39 of the 40 students are given in Table 2-1 (data was not available from one student).

In addition, an aide, or administrative assistant, was hired at each university. This aide was responsible for distributing all mail messages from the RTI coordinator to students, for filtering the general from the site-specific mail from students and forwarding it to the RTI coordinator or professor in charge, as appropriate, and for maintaining a hard copy notebook of the messages for student use.

2.5. The Development Environment

2.5.1. Introduction

Consistency in the software development environment across the universities involved in this study was necessary to allow comparisons of versions produced at each university. The environment was held as constant as possible by requiring each school to make available the same general development environment and by forbidding any one school to use any on-line design and development aids unless they were available at all schools.

2.5.2. On-line Environment

A Digital Equipment Corporation VAX 750 computer and the VAX 750 implementation of the Berkeley UNIX 4.2 BSD operating system were used at all sites. These were chosen because they were available at all four universities and most students were familiar, at least to some degree, with the operating system. (In fact, as shown in Table 2-1, 29 of 39 students listed UNIX as their favorite operating system.)

Each student was given his/her own account. Team members were put in the same UNIX "group," and read/write/execute access was within group only, for all participant accounts. This was one measure taken to insure independent development of versions across teams.

The C shell was chosen as the default shell for all student accounts because of its flexibility, history, and simpler script syntax. Note that nothing was done to prevent students from changing to another shell. The purpose was to present the students with the friendliest environment; if a student was more familiar with another shell,

Person	CS Graduate Course Hours	CS Undergrad Course Hours	Years CS Work Experience		Favorite Language	Favorite OS
			Full-Time	Part-Time		
			NCSU			
1	33	6	0	2.5	Pascal	VM/CMS
2	21	15	0	2	C	UNIX
3	0	10	0	0	Pascal	P-System
4	3	15	0	0	Pascal	VMS
5	9	12	0	1	Pascal	VMS
6	21	0	0	1	C	UNIX
7	6	6	0	0	Pascal	P-System
8	0	37	0	0	Pascal	CMS
9	9	27	0	0	Pascal	CMS
10	6	5	0	0	Pascal	CMS
			UIUC			
11	6	27	2	0	C	UNIX
12	6	40	1	0	Pascal	UNIX
13	18	70	0	0	Pascal	UNIX
14	9	15	0	0	Pascal	UNIX
15	48	12	0	5	Pascal	UNIX
16	33	45	0	2	Pascal	UNIX
17	48	12	.5	7	Pascal	UNIX
18	48	45	0	5	APL	UNIX
19	7	51	0	1	Pascal	UNIX
20	20	0	0	0	Pascal	UNIX
			UVA			
21	30	15	0	2	Forth	UNIX
22	9	0	1	5	Pascal	UNIX
23	27	0	0	0	Pascal	UNIX
24	6	15	.5	0	Pascal	UNIX
25	0	40	0	1	Pascal	UNIX
26	0	9	0	2	--	--
27	3	23	1	2	--	--
28	15	12	1	0	Pascal	UNIX
29	12	26	0	0	Pascal	UNIX
30	--	data	not	available	--	
			UCLA			
31	8	24	0	0	Pascal	UNIX
32	90	100	0	10	Algol	UNIX
33	24	16	0	5	C	UNIX
34	70	18	0	0	Pascal	UNIX
35	4	190	0	0	C	UNIX
36	38	0	.25	2	C	UNIX
37	42	[illegible]	1	2	Pascal	UNIX
38	12	40	0	0	Pascal	UNIX
39	50	4	.25	0	Pascal	UNIX
40	24	60	0	.5	C	UNIX

Table 2-1. Programmer Profile

he/she could change to it.

All RSDIMU procedures were written in Pascal. Not all compilers at all schools enforced the International Standards Organization (ISO) standards, but notice was given to all students that they should write their code in compliance with the standards. (This was required to increase portability of the resulting procedures, not to increase code quality or any other factor.)

The *vi* text editor was suggested as the editor of choice, and documentation on its features was provided. Nothing kept students from using other editors. The SDB symbolic debugger was made available although it was noted that bugs existed in the tool.

As mentioned above, only on-line tools available at all universities were allowed. For example, a mathematics package that would have helped students test matrix operations was available at only one school, and licensing requirements prohibited its distribution to all schools. Thus, while it would have been a helpful tool in this application, it was disallowed to keep a consistent environment across schools.

Students were told which terminals and printers were available for use at their school and were provided with information on their operation.

2.5.3. General Environment

Introductory training materials were prepared for each student on the elements of the on-line environment discussed above to insure all students had access to a minimum of information necessary to use the system. Students were required to work 40 hours per week, but working hours were flexible. Team mates were instructed to decide upon their work schedules together and to allow sufficient overlap for planning and discussions.

2.5.4. Communications Protocol

In anticipation of students' specification and other code development-related questions, a communication protocol was set up. A UNIX account was set up for the RTI coordinator, and that mail address was given to the aide hired at each university. Students were instructed to discuss problems with their teammate and, if the two of them could not solve it, to electronically mail the question to their aide. The aides' duties included receiving electronically-mailed questions from students at his/her school and filtering them as to whether they were site-specific or general. Any site-specific questions were to be solved on site; the aide was responsible for mailing the answer to the questioning team. Any general questions were forwarded to the RTI coordinator daily. The RTI coordinator processed the questions daily and distributed both the questions and their answers to the aides at all four universities, who in turn forwarded them to each team. For the students' use, each aide also kept a hard copy file of all questions and answers in chronological order by school.

3. Task Conduct

3.1. Time Schedule

The application was tailored to a size and complexity judged appropriate for teams of two students to design, code, and test in a ten-week period. This period was formally divided into time blocks for the design, code, and test phases with written documents due at predefined times. This was done both to mirror practices in industry and to allow us to monitor progress of all the teams. The time schedule with deliverables' due dates is shown in Figure 3-1. Each phase is further described below.

STAGE	TIME ALLOTTED	DELIVERABLES
Upon hire	N/A	Software engineering questionnaire
Design	4 weeks	Detailed design document Design walk-through report Weekly progress reports Daily time sheets
Code	2 weeks	Code development plan Unit tested, documented code Integration test plan Code walk-through document Weekly progress reports Daily time sheets
Integration test	2 weeks	Validation test log Weekly progress reports Daily time sheets
Acceptance Test	2 weeks	Acceptance test log Source code of passed program Post-experiment questionnaire Weekly progress reports Daily time sheets

Figure 3-1. Program Development Schedule

3.2. Design Phase

The four-week design phase began with assignment of students into teams, explanation of protocol, and distribution of training materials and RSDIMU specifications. As students became familiar with the specifications, electronic question-and-answer sessions began between students and the RTI coordinators. Students prepared preliminary designs, conducted formal design walk-throughs, and prepared reports of the problems found. A final, detailed design document was the primary output of this phase. In addition, students kept daily time logs and turned in weekly progress reports.

3.3. Code and Unit Test Phase

Two weeks were allowed for coding and unit testing. First, team members decided on the breakdown of code responsibilities; protocol required that one person code a unit and the other person test it. Students then coded, held a code walk-through and produced a report of its results, and tested their code. In addition, they completed an integration test plan for use in the next phase. The main deliverable was documented, unit-tested source code listings. Students also kept daily time logs and turned in weekly progress reports. Specification question-and-answer sessions continued throughout this phase.

3.4. Integration Test Phase

Two weeks were allowed for this phase. Students integrated and tested their code according to their plans and kept a validation test log of the process. In addition, they kept daily time logs and handed in weekly progress reports. The programs delivered at the end of this phase were considered ready for the "out-of-house" acceptance test. Specification question-and-answer sessions waned during this phase.

3.5. Acceptance Test Phase

The acceptance test in this phase subjected the versions to a minimum-requirements screening process. It insured that programmers had unit-tested and integration-tested their code and had followed coding standards prescribed.

Each university conducted its own acceptance testing. The test harness and input data sets were electronically mailed to each university, and each professor was in charge of running the acceptance tests for their teams' versions and distributing error data to the teams.

3.5.1. Input Distribution

Each test case in the suite of one million cases was generated from a random seed. The test case generator created data simulating noisy sensors from a realistic input distribution. In particular, the data were generated from the following domain description, which was provided by Charles River Analytics:

OBASE	15 inches
OFFRAW	For a given vehicle attitude (e.g., plane parked on a runway) simulate count data according to specified semioctahedron geometry for the eight channels.
LINSTD	3 counts
LINFAILIN	LINFAILIN <= LINFAILOUT
RAWLIN	The primary output of the reverse algorithm.
TEMP	Temperature on face A, B, C, D is: T_0+T_i, where T_0 is from a normal distribution with mean of 0 and standard deviation of 2° centigrade.
SCALE0	3.566
SCALE1	1.3585^{-2}
SCALE2	2.7169^{-5}
MISALIGN	Each misalignment angle is generated from a normal distribution with mean of 0 and a standard deviation of 0.10 milrads.
NORMFACE	A primary output of the reverse algorithm.
PHIV,THETAV,PSIV	Randomly chosen from 0 to 2π radians, within the constraints of the semioctahedron geometry.
PHII,THETAI,PSII	Randomly chosen from 0 to 2π radians, within the constraints of the semioctahedron geometry.
DMODE	Randomly chosen integer in the range (0...99).
LINOFFSET	Randomly chosen from a normal distribution with mean of 0 and standard deviation of 0.10.
DISMODE	Randomly chosen integer in the range (0...99).
BESTEST	Randomly chosen accelerations in the range (-g/4.0 ... +g/4.0).

3.5.2. Input data

Each team's test data was placed in one file, which we will refer to as a "test suite." Each test suite consisted of 25 fixed data sets plus 50 randomly generated data sets.

The 25 fixed sets were randomly generated and then hand-modified to simulate noisy sensor data. These 25 fixed sets simulated from 0 to 4 sensor failures during the calibration phase, and of these sets, approximately half also simulated an in-flight sensor failure. Thus, each of the sets has between 0 and 5 sensor failures per execution. Only one group of 25 sets was generated; therefore, the same 25 fixed sets were included in each team's test suite.

The 50 random sets simulated only noiseless sensors; thus, they included neither in-flight nor calibration-phase sensor failures. A set of 50 noiseless cases was randomly generated from a different seed for each team, so teams did not receive identical sets of

noiseless cases.

After distribution, some test cases were found to contain output variable errors. When this occurred, corrected test cases were generated and distributed to all universities along with instructions and explanations of the corrections.

3.5.3. Acceptance Test Harness

Student RSDIMU versions were subjected to an acceptance test; until a team's version passed this test, students on the team were required to continue to improve upon the reliability of their procedure through testing and debugging. The acceptance test of student programs insured that the code met portability standards outlined in the specifications and that final outputs agreed with the known outputs to within a set tolerance. The acceptance test harness was implemented in Pascal and was distributed to professors at each site.

The harness used an individual RSDIMU specimen, which was specified on the command line, and attempted to run it through its assigned test suite of 75 test cases. If outputs agreed on one data set, the harness looped and called the next data set in the suite. If at any point in the test-suite execution the RSDIMU outputs and correct outputs disagreed, the harness wrote the inputs, RSDIMU outputs, and correct outputs to a file and halted.

The harness also tested for portability and suitability for an N-version harness by checking to see if a student's RSDIMU made changes to input global variables, declared variables to be of type REAL or INT instead of the specified, portable types IREAL and IINT, contained case-sensitive code, called the voters in an order other than that called for in the specifications, or contained any underscores. Any RSDIMU procedures that had any of these above qualities was not passed through the acceptance test.

For test cases simulating noisy sensors, only the LINFAILOUT vector was checked. For test cases simulating noiseless sensors, LINOUT and BESTEST.ACCELERATION values were also checked. A conservative value of epsilon was used in comparing real number outputs for this phase. A student RSDIMU's value for LINOUT differing by more than 1 count from the precomputed value was failed, a value for a BESTEST.ACCELERATION component differing by more than 0.01 from the known value was failed, and a value in the boolean vector LINFAILOUT disagreeing with the precomputed value was also failed.

3.5.4. Acceptance Test Protocol

Before the two-week, acceptance-test phase, tapes containing the harness and test suites were sent to each professor. Each professor's aide arbitrarily matched up teams with test suites and ran a team's RSDIMU in the test harness with their assigned test suite when that team said their procedure was integration-tested. When the RSDIMU did not pass the acceptance test, the aide electronically mailed the harness output file to the team that created that RSDIMU, and the team fixed their procedure and advised the aide by mail when this had been done. The aide then ran the fixed

procedure through the acceptance test again using the *same* test suite.

There was no limit on the number of times a program might be subjected to acceptance testing; it was run through the harness and fixed until it passed all assigned input cases in the test suite. Thus, upon passing the acceptance test phase, all the RSDIMUs had been subjected to exactly 75 test cases.

4. Data

4.1. Descriptive Data

4.1.1. Subjects

Refer to Table 2-1 for descriptive information on the student programmers. These data were gathered from a questionnaire completed prior to the code design and development.

4.1.2. Specification Question-and-Answer Sessions

RTI coordinators received and answered a total of 242 questions from the students at the four universities. NCSU students submitted 79, UCLA submitted 35, UIUC submitted 66, and UVA submitted 62. Noting that UCLA students began work two weeks after the other schools and had volumes of answered questions awaiting them upon startup, it would be expected that those students would have submitted less questions. Taking this into account, no one school clearly outweighed another in the volume of questions posed.

In addition to the questions submitted by the students, a sequence of ten announcements were broadcast to all students; these announcements were prepared in response to the most controversial issues raised in the students' questions and were offered as the final answer on the issue.

The percentage of questions falling in different categories is given in Table 4-1, both on a per school basis and summed over all schools. The assignment of some questions into categories is clear cut, while assignment of others is, of necessity, subjective to a degree. For instance, distinguishing "design" questions was difficult since the specifications transcended functional specifications and included some design information. The point at which some questions departed from the design information in the specification was not always clear; only questions concerning design aspects that were not included in the specifications belong in the "design" category. Questions concerning aspects of design that were covered to some degree in the specifications belong in the most appropriate category other than "design."

It must also be noted that several questions, originating from one or more schools, were essentially rewordings of the same question. Because of the need to keep integrity in the Q & A numbering system, redundant questions were answered and were counted as if they were unique, individual questions in this discussion. This problem arose mainly from the geographical distribution of schools and time lag from one person's posing of the question to the time it appeared with an answer in each

	Clarification of Ambiguities	Specification Errors	Design	Previous Questions	Acceptance Test Cases	Nonexistent Specification Errors	Other	Total
NCSU	35	10	9	9	4	1	11	79 (33%)
UCLA	6	3	3	6	2	0	15	35 (14%)
UIUC	27	7	4	6	0	2	20	66 (27%)
UVA	19	10	8	1	2	2	20	62 (26%)
TOTAL	87 (36%)	30 (12%)	24 (10%)	22 (9%)	8 (3%)	5 (2%)	67 (28%)	242 (100%)

Table 4-1. Partitioning of Student Questions

student's electronic mailbox.

That the category receiving the largest percentage of questions (36%) was *Specification Ambiguities* shows that the English specifications were subject to differing connotations and required rewording, clarification, and expansion of some points. The questions in the next most common category, *Other*, with 28% indicated either a lack of knowledge in needed numerical and mathematical methods, or they showed a lack of study of the specifications prior to posing the question, as the answers were in the specifications. *Specification Errors* accounted for 12% of the questions; these errors ranged from typographical errors in spelling and equations to errors of omission of needed information. *Design* questions, as defined above, accounted for 10% of the questions. Design help was not provided in the answers to these questions. References to *Previous Questions*, for which the answer given was not accepted, accounted for 9% of the total questions, while questions concerning the *Acceptance Test Cases* took up 3% of the mail volume, and questions pointing out *Nonexistent Errors* in the specifications were the least common and accounted for only 2% of the questions.

It is interesting to note that almost half (48%) of the questions revealed ambiguities and errors. Since these problems were found by different teams and across universities, the N-version environment contributed to the debugging of the specifications. It is not possible to quantify from this task the potential benefits of N-version programming in specification debugging. Other experimenters [5,7] have postulated that N-version programming is better at revealing and masking programming faults than specification faults.

4.1.3. Versions

As shown in Table 4-2, source files ranged in size from 1,673 to 4,836 lines. The largest source file contained 78% blank and comment lines, while the smallest contained 38%, so a more detailed description is noteworthy. The average number of lines of code per version is 1,409; the range is from 963 to 2,296, and the sample standard deviation is 401. In terms of code tokens, the average is 10,516; the range is from 6,825 to 18,837, and the sample standard deviation is 3,692. The mean number of procedures and functions is 51 with a range from 29 to 90 and a sample standard deviation of 14.

Thus, the versions produced were structurally diverse. The versions were not analyzed for diversity of algorithms and numerical methods; this is recommended for future work.

RSDIMU VERSION	LINES IN SOURCE FILE	WHITE & COMMENT LINES	LINES OF CODE	TOKENS OF CODE	NUMBER OF PROCEDURES	NUMBER OF FUNCTIONS	TOTAL PROCEDURES & FUNCTIONS
A	4836**	3765**	1071	7266	43	11	54
B	3462	2472	990	6877	25*	7	32
C	1998	1035	963*	6825*	31	6	37
D	2336	1115	1221	9023	39	5	44
E	2059	806	1253	9634	39	11	50
F	3120	1113	2007	18003	43	2	45
G	3114	1530	1584	11810	81**	9	90**
H	2792	496	2296**	18837**	58	5	63
I	1819	475	1344	9362	44	10	54
J	2636	525	2111	17880	39	2	41
K	4121	2126	1995	12776	48	13	61
L	1807	727	1080	7934	25*	8	33
M	4037	2457	1580	11320	64	1*	65
N	1705	562	1143	7269	30	8	38
O	2310	838	1472	9404	52	2	54
P	1956	725	1231	8849	34	15**	49
Q	1687	418*	1269	10433	47	8	55
R	1874	756	1118	8359	48	8	56
S	2596	1179	1417	9945	63	2	65
T	1673*	644	1029	8505	27	2	29*

KEY

* lowest in category
** highest in category

Table 4-2. RSDIMU Version Structure

4.2. Empirical Data

Eighteen of the twenty versions were run through one million test cases. (Two versions were eliminated from this phase because after fairly extensive execution each abended as a result of an unhandled case in a case statement.) This data collection took approximately 170 CPU hours on RTI's Gould Power Node 9050 computing system. Given that one CPU hour on the Gould is equivalent to about seven CPU hours on a DEC VAX11/780, this was an intensive computing endeavor.

Each test case in the suite of one million cases was generated from a random seed. The test case generator created data simulating noisy sensors and other inputs from the input domain. As with the acceptance test harness noisy data cases, from 0 to 5 sensor failures were simulated per input case. Thus, life testing was not simply random; input values were chosen from the input domain, and at least one sensor failure occurred in approximately 74% of the test cases. The result is a random testing modified to stress test the code on sensor failures. Duran and Ntafos[8] point out that although random testing in many cases results in very good code coverage, untested branches seem to be mostly error-handling branches. They conclude that enhancing random testing with stress cases may result in a higher coverage.

For these reasons, testing was not strictly with an operational input series, but included a higher-than-operational percentage of stressful tests in an effort to simulate highly unlikely but possible circumstances under which software would execute a rare path and encounter a serious fault. Thus, we expect that measures of version and system reliability from these data would tend to be conservative, if anything.

These raw data are too voluminous to include in this report. They were written to twenty tapes via the UNIX operating system with data in binary form at 1600 bpi density on 2400-foot magnetic tapes. The information stored is version output for the variables: bestest.acceleration[3], linfailout[8], linout[8], and sysstatus.

5. Data Analysis

5.1. Preliminary Analyses

Preliminary data analyses were performed on a subset of the data to better understand the quality of the versions obtained. Full data analysis was outside the scope of this task.

5.1.1. Final Acceleration Estimates

The acceptance test checked the final acceleration outputs for non-noisy sensor data only to within 0.01 of the precomputed values. Post-acceptance testing compared the final acceleration outputs for noisy and noiseless sensor data, first between a test version and the precomputed values, and then across versions.

At best, versions can agree to within one part in 2^{12} of the precomputed linear acceleration values. This epsilon represents the best accuracy maintainable by an RSDIMU procedure. The primary input is the variable RAWLIN, which is a nonnegative number in the range of 0 to 2048; 12 bits are sufficient to represent a value of

RAWLIN. The primary outputs, however, are the values of BESTEST.ACCELERATION, which are real numbers and occupy 32 bits on the Gould PN9050. With perfect numerical methods, only 12 bits of this real number can be significant given the inputs from which it was calculated.

Using this definition of epsilon, the degree to which the versions agreed with the intended golden version's values depended largely on the absolute magnitude of the answers. Fractional values between -1 m/sec^2 and 1 m/sec^2 generally agreed less closely than estimates with larger absolute magnitudes. This is due to the change from an *absolute* epsilon, as used during acceptance testing, to a *relative* measure, as used in post-acceptance testing. Comparison among versions to within one part in 2^6 were then made, with this greater tolerance allowing for loss of accuracy during numerical computations. Versions failed often by this definition also; however, versions agreed closely among themselves on final acceleration estimates much more often than they did with the intended golden reverse solution. Thus, further version-to-gold comparative analyses were not performed. The intended golden solution must first be subjected to more extensive testing and numerical analysis.

Preliminary comparison of versions' acceleration estimates among themselves shows closer agreement across the majority of the versions than between the reverse-algorithm outputs and a version's data. Kelly and Avizienis[5] report a clustering and subclustering of real outputs among the versions. An analysis of the different algorithms used by different versions and their effects on the outputs is a possible area for future work; comparison with the algorithms and numerical methods used by the reverse algorithm would also expose differences across them.

5.1.2. System Status

Acceptance testing had checked only for correct designation of sensors as failed or operational when given noisy sensor data input. To see if programs correctly translated the sensor failures into the boolean designation of the system as operational or nonoperational, the SYSSTATUS output variable was considered in post-acceptance testing. Failure of a version in this discussion is defined by the version's disagreement with the majority of the versions on the boolean output SYSSTATUS.

Fifteen of the eighteen versions always agreed on the operational status of the system, over 25,000 test cases, as given in the boolean output variable SYSSTATUS. The three versions which failed (versions H, T, and P) were each from different universities. The fifteen versions in agreement reported that almost 65% of the test cases represent operational systems, and slightly over 35% represent nonoperational systems.

Version H always reported an operational system, and therefore fails on 35% of the tests. Version T always reported system failure; thus, the version fails on SYSSTATUS over 65% of the test cases. Version P reported system failure incorrectly on 25% of the test cases (see Table 5-1). Therefore, correct determination of sensor failures by the procedures did not, in these three versions, map to correct determination of system operational status. A fault analysis of the versions exhibiting these failures could be performed to provide additional insight into the failures.

	VERSION							
MAJORITY	H		P		T		ALL OTHERS	
	True	False	True	False	True	False	True	False
TRUE	16,229	0	10,087	6,142	0	16,229	16,229	0
FALSE	8,771	0	0	8,771	0	8,771	0	8,771

Table 5-1. SYSSTATUS Results on 25,000 Test Cases

5.2. Discussion

The difficulty of finding an appropriate methodology to determine correctness by voting among real number outputs of N-version codes has long been recognized.[9,10] The task is complicated in this application because of several factors: interactions among several input variables determine the acceleration outputs (the mapping of input space to output space is not simple); binary decisions concerning sensor health (operational/failed) are made based on results of computations with real numbers—a slight difference in real number values of data or of a version's decision cutoff value may sway the decision and all remaining computations; and there is no history of sensor readings and computations in this implementation, thus, there can be no time-averaging of values to aid in making boolean filtering decisions as there often are in real-time process controls.

An analytical determination of epsilon is arbitrary at best. A previous study has shown that an increase in range of tolerance may increase the rate of non-detection of actual failures, and a smaller range may increase the number of false failures reported.[11] Until further work is performed to define the correctness of real number outputs of this problem in a meaningful way, single version and N-version system reliability or failure probability estimates cannot be made on the basis of real number outputs, and neither can data be obtained for use in Eckhardt and Lee's coincident error model. In future specifications for N-version development, stating the precision and accuracy requirements of floating point outputs is strongly recommended.

6. Conclusions

6.1. Other Data Analyses

The scope of this task did not call for extended data analyses. Analyses that can be performed on the data gathered here are outlined below.

6.1.1. Error Analysis

After extensive testing of an oracle, or golden version, of the RSDIMU procedure and determining the real number accuracy required, the million-test-case data can be analyzed. The primary analysis recommended for use is Eckhardt and Lee's coincident error model to model the intensity of coincidentally occurring errors in the final acceleration estimate and to show the optimal number of versions for the N-version system constructed from the available versions.

Cluster analyses could be performed on the linear acceleration data collected. This analysis does not require production of an extensively tested golden version, but time and effort would be required to design an appropriate clustering methodology and analysis.

6.1.2. Fault Analysis

In an N-version framework, coincidentally occurring errors in different versions subvert the fault-detection capabilities of the system. Discovering ways of reducing coincident errors is therefore of primary importance for ultra-reliable N-version systems. Analysis of faults in the versions developed could bring insight into the types and locations of faults leading to coincident errors.

An important question is whether the coincident errors are caused by the same or different faults across versions. If, by the same faults, in what phase of the software life cycle were these faults introduced? Can they be traced to a specification error, to the inherent difficulty of an algorithm, or to a defective development environment, such as a mathematical/statistical package with one or more bugs? If coincident errors that are found are caused by different faults, are the faults in the same algorithm or same logical portion of code?

Another important question concerns the faults resulting in noncoincident errors. What is the nature of these faults, and in what phase of the life cycle were they introduced? Are designs of versions with and without the fault diverse? Does fault masking occur such that in a program with noncoincident failures, when one of a number of faults is corrected, a remaining fault now results in errors coincident with other versions?

In this task, we did not inspect code to locate the faults which caused errors in the versions. An examination of the faults in the various versions and a comparison of the faults across versions to see if common mistakes were made may provide insight into software bug prevention/elimination. A further analysis could attempt to find the cause of the faults; that is, were the requirements specifications in error, ambiguous, or incomplete regarding the concept on which the fault occurred?

6.2. Other Studies

If a lack of design diversity in independently developed versions results in faults that are manifested in coincident errors, enforcement of design diversity[12] may be a means of increasing N-version system reliability. Studies can be performed on existing

independently developed N-version codes to qualitatively assess design diversity, to identify faults that are manifested in coincident errors, and to determine if faults across versions are related. If related design faults cause coincident errors, forcing design diversity may reduce error coincidence.

Another potential benefit of controlling the development of different designs is that all designs can be screened via numerical analyses prior to implementation to insure they produce results that are amenable to a reasonable voting scheme. This would result in less dynamic modeling to fine tune tolerances of voters once the N-version system is assembled. It is not certain that forcing diverse design is practical in all applications; a potential problem is that diverse designs may preclude outputs falling in as narrow a range of tolerance as required by the specifications as a result of a variance in the accuracy attainable by different methods used. That is, there may be applications for which there is clearly one best way to design the solution, and other designs are poorer approximations of the solution.

Another question concerns the cost/benefit ratio of enforcing design diversity. If forced diversity does result in negatively correlated errors, is the increase in system reliability obtained at an increased cost or at a savings? Will forcing diversity mean that less versions will have to be developed to achieve reliability goals, thus, saving money? If so, will the savings by developing less versions be offset by the money spent in analyzing and choosing appropriate diverse designs? Case studies can be performed to begin to answer these questions concerning the option of enforcing diversity at the design level.

To study the effects of N-version versus single-version development on the debugging of specifications, case studies could also be performed. The subjects working in the N-version environment would partake in the controlled communications involving specification question-and-answer sessions as were provided in this task; single-version developers would only receive answers to questions they explicitly asked. Time and development cost allowed for both groups could be held at a constant so cost/time/system reliability analyses could also be performed. It has been speculated[5,7] that N-version development is better at finding code bugs than specification bugs. A controlled study would provide more evidence as to whether this is actually the case.

6.3. Design Improvements

Valuable experience gained from this work can be applied to planning for software specification, design, and development in future code development efforts for software experiments.

6.3.1. Reduce Geographic Separation of Sites

While wide geographic separation of developmental sites may lend to the independent design and development of software, it hindered progress in both the design and code development phases. Meetings at a location and time amenable to all principal

investigators' schedules were difficult to plan; thus, design meetings were fewer than hoped for and were delayed because of schedule conflicts.

6.3.2. Complete Experimental Tools Early

The biggest drawback from design delays was that there was not enough time for specification debugging and tool implementation/testing prior to the software development phase. Because of this schedule slippage and the inflexibility in conduct dates (subjects were graduate students on summer vacation), our timetable did not allow for advance preparation and extensive testing of a reverse algorithm golden version or prototype version of the application or for completion of all software tools. It is now clear that advance preparation of all tools used in studies of this type should be required.

6.3.3. Conduct Trial Run

The time spent in having one or two programmers follow the proposed design and development protocol to produce a version of the application under study would have saved time during and following the actual development by: subjecting the specifications to detailed scrutiny for debugging prior to release to subjects; providing "experts" with in-depth knowledge of the design and implementation process; relieving the need to develop an inverse algorithm for use as test case generator; and providing a sample version to use in testing the acceptance test harness and life test harness. These points are elucidated below.

While debugging specifications prior to production are not a standard industry practice, they are almost necessary in multi-site software development efforts. This is due to the time lag in receiving questions from diverse sites and returning answers to all sites. The on-line mail network was the most efficient communications medium available, yet in-transit time was often longer from RTI to California and Illinois than to North Carolina and Virginia as a result of the different number of intervening communication nodes from site to site. The volume of mail traffic, as well as the time differential in arrival times based on size of messages, also had the undesirable side effect of causing confusion as to the chronological order of specification updates; extra care had to be taken to insure that everyone knew which updates superceded which. Had the specifications been subjected to the degree of scrutiny only design and implementation can provide, an estimated 50% of the questions received from the students could have been avoided. A larger percentage likely could have been avoided, also, had aeronautical engineers been employed as programmers. In this case it would not have been necessary to expand the specifications to include sections outlining mathematical methods, and we expect less ambiguities and misunderstandings of the problem would have occurred. This would have reduced both subjects' and coordinator's time spent on this problem during the program development stage.

The knowledge base acquired by having these programmers actually step through the same process as the students may have been invaluable both in validating the design and protocol and in producing "technical experts" who could act as consultants and answer specifications questions arising during the design and development.

Since the algorithm could be implemented in reverse to provide a test case generator, a golden version was not essential for providing test cases for the students. However, the most critical inputs to the reverse algorithm are real numbers, and its most critical outputs are integers. Thus, the reverse is true of the forward algorithm. This led to the need for an in-depth look at a numerical analysis of the problem to determine real number tolerance to allow for when checking correctness of results of the RSDIMU procedures against the inputs to the reverse algorithm.

A golden version and controlled mutations of it would have provided a useful tool for validating that the acceptance test harness correctly trapped nonstandard Pascal constructs and erroneous outputs and that the life test harness ran correctly on test suites of correct and incorrect RSDIMU outputs.

6.3.4. Improve Development Environment

Another issue in the task design concerns the software development environment. In this task, we used the UNIX 4.2 BSD operating system and Berkeley Pascal because these tools were available at all universities. UNIX is a powerful research tool and is unique in that UNIX sites receive and may alter the source code of the operating system. UNIX Pascal compilers may also be customized and produce different executables. Thus, there is no way to insure consistency in the on-line environment of subjects using UNIX at one site with the environment of subjects using the same version of UNIX at another site.

The differences in fault types and numbers, if any, are not known, from avionics software written in Pascal at different sites by graduate students and targeted for a UNIX system, to that written by avionics programmers in other languages and targeted for specialized hardware.

Also, an on-site visit at one of the participating universities highlighted logistical problems resulting from lack of physical space for students. In this task it would have been desirable to give team members proximity to each other, yet physical separation from other teams, as well as separation of all teams from the on-site aide. Lack of space precluded implementing these ideal working conditions to insure independent development of versions.

Computing facilities were heavily used at the visited site. High load factors and resultant slow response time were observed. This was a hindrance to progress in the code implementation and test phases, as well as in on-line document preparation in all phases. It is unknown how much the load factors varied among the sites or how much, if any, effect this factor had on resulting quality of versions.

6.4. Summary

This work has added to the knowledge base to benefit future software experimentation. It has highlighted time-critical and resource-critical areas in early planning and development phases that strongly influence the entire experiment. Generally, time spent in developing and testing supplementary tools and procedures in advance will

pay off, as less time will be required in these areas during the actual code development phase. This is especially important if the code development phase is the phase with the least flexible time schedule, as in the case when graduate students are employed for the summer.

Ample resources and time to iteratively write and review problem specifications can reduce errors, ambiguities, and omissions in the specifications delivered to the subjects. This preparation will save time which would otherwise be spent later processing subjects' questions. Numerical considerations should be fully addressed in the problem specification, as well as in the specifications for software tools, such as the N-version voting procedure. Identifying, developing, and testing all software tools prior to the program development stage are also critical to smooth operation of that phase.

In the design stage, issues associated with multiple, geographically separated code development sites must be addressed and resolved. This experience has shown that the wide geographic separation among principal investigators hindered progress in the planning phase; the separation of subjects hindered communications and progress in the execution phase; and the diversity of UNIX computing environments hindered the production of code that will meet portability requirements.

A trial run through the code development process by one or two programmers is recommended, as this will bring deficiencies in the protocol to the surface, serve to debug the specifications and software tools, and provide the programmers with the background to serve as knowledgeable consultants during the actual development stage.

In planning code development for future large-scale studies in software fault tolerance, these issues and others discussed in the body of this report should be given full consideration.

References

1. Algirdas Avizienis, "Fault-Tolerance: The Survival Attribute of Digital Systems," *Proceedings of the IEEE* **66**(12) pp. 1109-1125 IEEE, (October 1978).
2. J. C. Knight, Nancy G. Leveson, and Lois D. St. Jean, "A Large Scale Experiment in N-Version Programming," *FTCS 15th Annual International Symposium on Fault-Tolerant Computing*, pp. 135-139 IEEE, (June 1985).
3. J. C. Knight and N. G. Leveson, "An Experimental Evaluation of the Assumption of Independence in Multiversion Programming," *IEEE Transactions on Software Engineering* **Vol. SE-12, No. 1** pp. 96-109 IEEE, (January 1986).
4. Dave E. Eckhardt and Larry D. Lee, "A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors," *IEEE-TSE* **SE-11**(12) pp. 1511-1517 (DeC 1985).
5. J. P. J. Kelly and Algirdas Avizienis, "Multi-Version Software Development," *Proceedings of SAFECOMP '86*, Pergammon Press, (October 1986).
6. Janet R. Dunham, "Evaluating Software Dependability via Experimentation," *14th International Conference on Fault-Tolerant Computing*, IEEE, (June 1984).
7. John C. Knight and Nancy G. Leveson, "An Empirical Study of Failure Probabilities in Multi-Version Software," *FTCS 16th Annual International Symposium on Fault-Tolerant Computing*, pp. 165-170 IEEE, (July 1986).
8. Joe W. Duran and Simeon C. Ntafos, "An Evaluation of Random Testing," *IEEE Transactions on Software Engineering* **Vol. SE-10, No. 4** pp. 438-444 IEEE, (July 1984).
9. L. Chen and A. Avizienis, "N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation," *Digest of Papers, FTCS-8: Eighth Annual International Conference on Fault-Tolerant Computing*, pp. 3-9 IEEE, (June 1978).
10. T. Anderson and P. A. Lee, *Fault Tolerance: Principles and Practice*, Prentice Hall, London (1981).
11. Janet R. Dunham and John L. Pierce, "An Empirical Study of Flight Control Software Reliability," *NASA Contractor Report 178058*, (March 1986).
12. Douglas Miller, "A Mathematical Framework for Multiversion Software with Diverse Components," IFIP Working Group 10.4 - Reliable Computing and Fault Tolerance, Baden, Austria (June 26, 1986).

Appendix A. RSDIMU Specifications

REDUNDANCY MANAGEMENT SOFTWARE

REQUIREMENTS SPECIFICATION

FOR A

REDUNDANT STRAPPED DOWN INERTIAL MEASUREMENT UNIT

30 May 1985

Charles River Analytics

Research Triangle Institute

Version 2.0

Table of Contents

1 INTRODUCTION 31
1.1 General 31
1.2 Geometry 33
2 FUNCTIONAL REQUIREMENTS 36
2.1 General 37
2.2 Mappings 37
2.3 Input Variables 41
2.3.1 Geometry 41
2.3.2 Sensor Data 42
2.3.2.1 Failed Sensor Status 42
2.3.2.2 Raw Data 42
2.3.3 Calibration 44
2.3.3.1 Determination of Slope 44
2.3.3.2 Determination of Offset 45
2.3.3.3 Validation of Offset 46
2.3.4 Detection Threshold 47
2.3.5 Display Control 47
2.4 Output Variables 47
2.4.1 Calibration Outputs 48
2.4.2 Failed Sensor Outputs 48
2.4.3 Vehicle State Estimation 50
2.4.3.1 Linear Acceleration Outputs 50
2.4.3.2 Unsolvable System 50
2.4.3.3 Vehicle State Outputs 51
2.4.4 Output Panel Display 52
2.5 Sensor Failure Detection And Isolation 59
2.6 Vehicle Acceleration 62
3 PASCAL DECLARATIONS 64
3.1 Constants 64
3.2 Types 65
3.3 Variables 67
3.4 Voters 68
4 BIBLIOGRAPHY 70

5 APPENDIX A - PROBLEM COORDINATE FRAMES 71
5.1 Local Navigation Frame (N) 71
5.2 Vehicle Frame (V) 72
5.3 Instrument Frame (I) 76
5.4 Sensor Frames (A, B, C, D) 79
5.5 Measurement Frames 81
6 APPENDIX B - ACCELEROMETER MEASUREMENT EQUATIONS 84
6.1 Specific Force 84
6.2 Accelerometer Measurements 85
7 APPENDIX C - LEAST SQUARES VEHICLE ACCELERATION ESTIMATION 87
8 APPENDIX D - EDGE VECTOR TEST 89
8.1 Edge Vector Relations 89
8.2 Test Thresholds 89
8.3 Sensor Failure Detection And Isolation 90
9 APPENDIX E - SOFTWARE & HARDWARE SUPPORT 91
9.1 Software 91
9.1.1 Operating System 91
9.1.2 User Interface 91
9.1.3 Protection 91
9.1.4 Use of File System 91
9.1.5 Tools Set 91
9.1.5.1 Pascal 91
9.1.5.2 Editor 91
9.1.5.3 SDB 92
9.1.5.4 Version Control 92
9.1.5.5 Pascal Cross Reference and Pretty Printing 92
9.1.5.6 Gprof: Profile program 92
9.1.5.7 Configuration Control 92
9.1.5.8 Mail 92
9.1.5.9 Documentation Tools 92
9.2 Hardware 92
9.2.1 Computing Machinery 93
9.2.2 Terminals 93
9.2.3 Printers 93

1. INTRODUCTION

1.1. General

As part of an integrated avionics system, you are to develop a parameterless Pascal procedure to be called RSDIMU, for the management of sensor redundancy in a Redundant Strapped Down Inertial Measurement Unit (RSDIMU). An RSDIMU is used as part of the navigation system in aircraft and spacecraft.

An RSDIMU consists of a skewed array of redundant inertial sensors and exemplifies the current trend for designing hardware fault tolerant inertial measurement units (IMUs) for high reliability applications. The portion of the RSDIMU you will handle contains eight linear accelerometers mounted on the four triangular faces of a semioctahedron.

Each accelerometer measures *specific force* along its associated measurement axis, where specific force is the difference between the RSDIMU's inertial linear acceleration and the acceleration due to gravity. You are to process these sensor measurements to provide estimates of the linear acceleration of the vehicle in which the RSDIMU is installed.

Primary inputs to the procedure consist of the sensor measurement values from each of the eight accelerometers. Secondary inputs consist of information describing the problem geometry and system specifications.

Your procedure will have two functions, both of which are a consequence of the redundancy in the sensor complement of the RSDIMU. The first function is to per-

form a consistency check to detect and isolate failed sensors. The second is to use the sensors found to be good by the first check to provide estimates of the vehicle's linear acceleration expressed as components along the north, east, and down axes of a navigation frame of reference.

Primary outputs are a sensor status vector specifying either a failed or an operational mode, and a set of estimates for the vehicle's linear acceleration based on various subsets of the operational sensors. Secondary outputs drive a display panel and provide system status information. Your procedure will be used as one of several modules in a fault tolerant software system, and the sensor failure output will be passed to a voter which may alter its value prior to being used for the estimation of vehicle acceleration. Figure 1 is a block diagram of the procedure.

In practice, an RSDIMU as described here would operate as follows. With the vehicle stationary, a series of sensor readings would be taken over time, and this would comprise the calibration data set for that particular flight. During flight, the sensors would be read periodically at regular time intervals to provide input for the navigation software.

For the purposes of this problem, the input will include the calibration data set and a single set of sensor values taken at a single time during flight. Your procedure is to first perform calibration using the calibration data and to then process the single set of flight data.

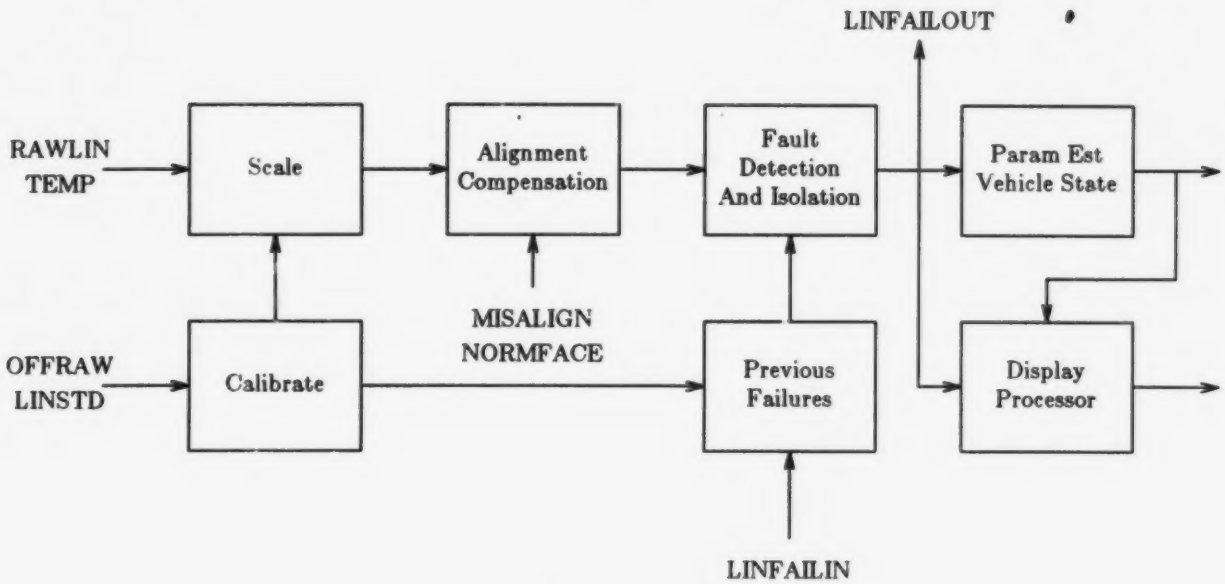


Figure 1. System Data Flow Diagram

1.2. Geometry

For purposes of this application, the geometric considerations have been somewhat simplified. The RSDIMU is assumed to be located at the vehicle center of gravity (CG). The instrument unit is assumed to be a perfect regular semioctahedron. However, sensors are not perfectly aligned on the unit. Under these circumstances, there are eleven coordinate systems of interest. Mathematical descriptions of these coordinate systems are given in Appendix A.

The first coordinate system is a local navigation frame, known as the *Navigation Frame of Reference* and designated by N in this document, which has its origin at a fixed location on the earth and has its X axis aligned with north, Y axis aligned with east, and Z axis aligned with *down* (see Figure A1). These axes will be labeled X_N, Y_N, Z_N where the axis Z_N is aligned with the local gravity vector. For the purpose of this problem, earth's rotation will be neglected, and the N frame will be considered to be inertial.

The second coordinate system is the vehicle body coordinate system, known as the *Vehicle Frame of Reference* and designated by V in this document (see Figure A2). Referring to Figure A2, the X axis points forward along the nose, the Y axis points out along the right wing, and the Z axis points down. These axes will be denoted as X_V, Y_V, and Z_V. The origin of the Vehicle Frame of Reference is the center of gravity (CG) of the vehicle. The relationship between the Vehicle Frame of Reference and the Navigation Frame of Reference is given in terms of three angles representing the orientation of the V frame with respect to the N frame (see Figures A2-A3). The angles ϕ_V, θ_V, and ψ_V represent the rotations which bring the N frame into coincidence with the V frame.

The sensors are mounted on the triangular faces of a semioctahedron. A *semioctahedron* is a pyramid with a square base and four sides, each of which is an equilateral triangle (see Figure A4). The remaining nine coordinate systems are defined with respect to this semioctahedron. The *Instrument* (I) *Frame of Reference* consists of three orthogonal axes X_I, Y_I, and Z_I. The X_I and Y_I axes are aligned with the diagonals

of the base of the semioctahedron, while Z_I is normal to the base and runs from the intersection of the base diagonals to the apex of the semioctahedron.

The relationship between the Vehicle Frame of Reference and the Instrument Frame of Reference is given in terms of three angles representing the orientation of the I frame with respect to the V frame. The origin of the I Frame is assumed to be the same as the origin of the V Frame. The angles ϕ_I, θ_I, and ψ_I represent the rotations which bring the V Frame of Reference into coincidence with the I Frame of Reference (see Figure A5).

On each triangular face of the semioctahedron is an orthogonal coordinate system associated with the two linear accelerometers. These four coordinate systems will be referred to as the *Sensor Frames of Reference* For convenience, we will label the faces and the corresponding Sensor Frames of Reference A, B, C, and D, with face A having its base in the quadrant of the X_I and Y_I plane in which both X and Y values are positive (see Figure A4). The labeling proceeds counterclockwise about the plane, i.e., face B has its base in the positive X and negative Y quadrant, etc. For face A, the origin of the Sensor Frame of Reference is located at the face centroid (see Figure A6). The Z_A axis is normal to the face and points out. The X_A and Y_A axes lie in the plane of the face, point to the semioctahedron base edge, and are symmetric about the face center, completing a ***right-handed*** orthogonal coordinate system in which counterclockwise rotation is positive. The Sensor Frames of Reference for faces B, C, and D are defined in analogous terms.

On each triangular face of the semioctahedron, there is an additional nonorthogonal coordinate system, associated with the set of two linear accelerometers on that face. These four coordinate systems, one associated with each face, will be referred to as Measurement Frames of Reference. They are labeled $\bar{A}$, $\bar{B}$, $\bar{C}$, and $\bar{D}$ in correspondence with the Sensor Frames of Reference.

The origin of each Measurement Frame is co-located with that for the corresponding Sensor Frame of Reference. The axes of each Measurement Frame are *nonorthogonal* but differ by only small angles[1] from the corresponding axes of the orthogonal Sensor Frame of Reference on the same face. These small angles will be denoted as misalignment angles. Accelerometers measure the projections of the specific force exerted on the RSDIMU onto the two axes that are misaligned from the face of the semioctahedron. These four nonorthogonal coordinate systems will be referred to as *Measurement Frames of Reference.* The misalignment between the two corresponding axes of the Sensor and Measurement Frame of Reference is defined by two angles, each of which represents an independent rotation about a particular axis of the orthogonal Sensor Frame of Reference (see Figure A7). Hence, there are four Accelerometer Measurement Frames of Reference, each defined by six different misalignment angles.

2. FUNCTIONAL REQUIREMENTS

[1]For our purposes, small angles are ones for which the approximation $\sin\theta = \theta$ is valid. The angles will be less than $\pm 5^\circ$ and typically in the range of less than $\pm 1^\circ$.

2.1. General

All communications between your procedure and its environment are through the variables and constants discussed in the sections entitled Input Variables and Output Variables below. Computational requirements are covered in a subsequent section. The mappings which follow apply to inputs and outputs in the case of sensor related items and to outputs only for the others. The input and output variables are to be made global to your procedure. Your procedure may not change the value of any input variable. Types, constants, and variables, along with the voting routines are supplied in the files consts.h, types.h, vars.h, and votes.h. Your procedure may not depend on any other global declarations for operation. Any additional real or integer types you require should be declared in terms of the types supplied in types.h. You may not directly use the built-in Pascal types "integer" and "real".

2.2. Mappings

Many inputs are provided as arrays. For all inputs which are related to sensors mounted on the faces for the semioctahedron, the mapping from array index to face axis is as follows:

Index	Axis
1	X_A
2	Y_A
3	X_B
4	Y_B
5	X_C
6	Y_C
7	X_D
8	Y_D

For face specific inputs, the mapping from index to face is as follows:

Index	Face
1	A
2	B
3	C
4	D

Misalignment angles represent the rotation of an axis in a Measurement Frame of Reference about an axis in the corresponding Sensor Frame of Reference. Let the notation XY, for example, stand for the rotation of Sensor Frame axis X about axis Y in the same frame. There are six such angles of interest, mapped as follows:

Index	Angle
1	XY
2	XZ
3	YX
4	YZ
5	ZX
6	ZY

The acceleration outputs consist of three linear acceleration components along the navigation axes X_N, Y_N, Z_N. In these arrays, the indices are mapped as follows:

Index	Axis
1	X_N
2	Y_N
3	Z_N

The display output consists of a three word array for each of the two five digit displays. In these arrays, the indices are mapped as follows:

Index	Element
1	Word1
2	Word2
3	Word3

Vehicle state is reported for four channels composed of two faces each. The mapping from channel index to face pairs when all faces are operational is as follows:

Channel	Faces
1	A, B
2	B, C
3	C, D
4	D, A

The mapping from channel index to face pairs when only three faces are operational (i.e. when both of the accelerometers on a given face are declared faulty) is as follows:

	Face A Failure	Face B Failure	Face C Failure	Face D Failure
Channel	Faces	Faces	Faces	Faces
1	—	A,C	A,B	A,B
2	B,C	—	B,D	B,C
3	C,D	C,D	—	C,A
4	D,B	D,A	D,A	—

The mapping from channel index to face pairs when only two faces are operational (i.e., when all four accelerometers on two faces are declared faulty) is as follows:

	A,B Failure	A,C Failure	A,D Failure	B,C Failure	B,D Failure	C,D Failure
Channel	Faces	Faces	Faces	Faces	Faces	Faces
1	—	—	—	—	A,C	A,B
2	—	B,D	B,C	—	—	—
3	C,D	—	—	—	—	—
4	—	—	—	D,A	—	—

In principle, any face pair may be assigned to any channel. As noted above, the channel/face pair assignments change as sensors and faces fail. The following mapping is used to record the assignment of face pairs to channels.

Value	Face Pair
0	Nonoperational
1	A,B
2	A,C
3	A,D
4	B,C
5	B,D
6	C,D

2.3. Input Variables

Inputs are divided into four categories: Geometry, Sensor Data, Calibration, and Display. These, along with the output variables, are to be considered global to the procedure. Declarations of these variables and their types follow the description.

2.3.1. Geometry

These variables define the geometry of the semioctahedron relationship between the N, V and I Frames of Reference. They remain constant for an input case, but may change between input cases.

OBASE	The length of one side of the square base of the semioctahedron. Units are inches.
PHIV THETAV PSIV	Rotation of the V Frame of Reference with respect to the N Frame of Reference. Rotations are defined for a yaw, pitch, roll Euler rotation sequence as defined in Appendix A. Units are degrees.
PHII THETAI PSII	Rotation of the I Frame of Reference with respect to the V Frame of Reference. Rotations are defined for a yaw, pitch, roll Euler rotation sequence as defined in Appendix A. Units are degrees.
MISALIGN(i,j)	Measurement Frame misalignment angles for face i (A, B, C, or D) with rotation j (XY, XZ, YX, YZ, ZX, ZY). Units are milrads.

In order to project the misaligned sensor data onto the idealized face coordinate systems, it is necessary to know the accelerations normal to each face, i.e., along the Z

axis of the face. In an actual system, this value is derived from the vehicle accelerations computed for the preceding time step in the time series of computations. For this problem, it is given in the following variables.

NORMFACE(i)	is the acceleration component normal to face i (i = 1 to 4) of the semioctahedron. Units are meters sec^{-2}.

2.3.2. Sensor Data

2.3.2.1. Failed Sensor Status

This is an array of Boolean values indicating which sensors are operational and which have failed prior to this invocation.

LINFAILIN(i)	LINFAILIN(i) = **TRUE** implies that linear accelerometer i, (i=1 to 8), has been identified as failed on some earlier invocation of the procedure. LINFAILIN(i) = **FALSE** implies that sensor i was found to be operational on the immediately preceding invocation of the procedure. Sensors marked as failed by this input are not to be considered in subsequent processing and should be noted as failed in the outputs.

2.3.2.2. Raw Data

Raw data is derived from linear accelerometers which provide outputs indicating the specific force along axes in a face of the semioctahedron platform. The accelerometers produce voltages which are converted into a digital input with an analog to digital converter. The output of this converter is the raw data input to the procedure. A

typical output appears as 12 bits representing an unsigned, positive integer in a 16-bit computer word as follows:

MSB															LSB
X	X	X	X	(				Indicated Value							)
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

In this notation, the (and) delimit the most and least significant bits of a named field in the word. An X indicates a bit which may contain either a zero or a one. The range of the conversion is *nominally* -5.0 volts to +5.0 volts; thus, scaling of the indicated value is as follows:

$$\text{Scale : } 4096/10 = 409.6 \text{ counts/volt}$$

$$\text{Offset : } 2048 \text{ counts} = 0.00 \text{ volts}$$

These factors will be used to determine the Voltage output of the accelerometer using the equation:

$$\text{Voltage} = (\text{IndicatedValue} - 2048)/409.6$$

The Voltage is in turn used to determine the acceleration using the calibration equations discussed below.

RAWLIN(i)	is the array of linear acceleration raw data values, (i = 1 to 8). These are raw data words as described above.

2.3.3. Calibration

These variables convert the raw data voltages from the linear accelerometers into engineering units and provide information about the noise spectra of the raw data.

Linear acceleration is given by the equation:

$$\text{LinearAcceleration} = \text{Offset} + \text{Slope} \times \text{Voltage}$$

The factors used in determining slope are given. The offset is determined by the calibration procedure given below.

2.3.3.1. Determination of Slope

Accelerometer slope is a function of scale factors which are temperature dependent and is to be computed by:

$$\text{Slope} = \text{Scale0} + \text{Scale1} \times \text{Temp} + \text{Scale2} \times \text{Temp}^2$$

where Scale0, Scale1, Scale2 are the accelerometer scale factor temperature sensitivity coefficients and Temp is the current operating temperature of a given face. The scale factors are determined in the laboratory before the RSDIMU is installed in the vehicle. This is done by subjecting the instrument to a series of known forces and temperatures and using a statistical analysis procedure to estimate the appropriate scale values. The values supplied to the procedure are the result of this process. The scale factors and temperature to be used are contained in the following variables:

TEMP(i)	is the temperature of the instrument on the i'th face (i = 1 to 4). It is given in degrees Celsius.

SCALE0(i) SCALE1(i) SCALE2(i)	give the coefficients of the slope equation for sensor i (i = 1 to 8). The units for SCALE0 are meters·sec^{-2}·volt^{-1}. The units for SCALE1 are meters·sec^{-2}·volts^{-1}·°c^{-1}. The units for SCALE2 are meters·sec^{-2}·volts^{-1}·°c^{-2}.

2.3.3.2. Determination of Offset

Offsets are also determined in the laboratory, but the nature of the accelerometer is such that its offset changes with time and must be redetermined at the start of each flight. We note that when the vehicle is at rest on the ground, prior to a flight, that the only force acting on the RSDIMU is the force of gravity, g. Under this circumstance, each accelerometer measures the component of g along its axis. For example, the accelerometer for the X axis of face A is measuring a force $g_{X_{\bar{A}}}$, the projection of g along the misaligned X_A axis. With the vehicle at rest, the Z_V and Z_N axes are assumed to be aligned so that only the rotation of the I frame with respect to the V frame, the relationships between the A and I frames, given by the geometry of the semioctahedron, and the misalignment of the $X_{I\bar{A}}$ axis need be considered in finding $g_{X_{I\bar{A}}}$.

Now let $\bar{I}_{X_{I\bar{A}}}$ be the average indicated value of the calibration values for the sensor for axis $X_{I\bar{A}}$. The corresponding voltage will be:

$$\bar{V}_{X_{I\bar{A}}} = (\bar{I}_{X_{I\bar{A}}} - 2048)*409.6$$

Inserting this in the calibration equation given above, we obtain:

$$g_{X_{iA}} = \text{Offset} + \text{Slope} \times \bar{V}_{X_{iA}}$$

which may be solved for the offset value. This value is used for transforming the subsequent flight data.

2.3.3.3. Validation of Offset

Accelerometer outputs are noisy. Their actual value can be viewed as the sum of the output of a perfect, noise-free sensor and a random variable with a zero mean and a given standard deviation. During the pre-flight computation of offset, this noise is compensated for by averaging. A properly functioning sensor has a known standard deviation for its noise component, given as the input variable LINSTD. By making a comparison between this value and the standard deviation of the calibration data used to compute the offset, it is possible to detect certain types of sensor failure. For our purposes, we will consider as failed, any sensor whose standard deviation, $S_{X_{iA}}$ is more than three times LINSTD.

$$S_{X_{iA}} = \sqrt{\sum_{i=1}^{n} (I_{X_{iA}}(i) - \bar{I}_{X_{iA}})^2 / n}$$

where n is the number of elements in the calibration array, $I_{X_{iA}}(i)$ represents an individual element of this array and $\bar{I}_{X_{iA}}$ is the average value of the array elements.

The input variables used for the offset calculation are given below. The results are part of the output from the procedure and appear in output variables discussed in a subsequent section.

OFFRAW(i,j)	is an array of elements in raw data format used to determine the offset for sensor i (i=1 to 8), (j=1 to CUB).
LINSTD	is the standard deviation of the noise for a properly functioning sensor. Its value is given in counts so that it may be directly compared with the indicated values of raw data.

2.3.4. Detection Threshold

This variable controls the sensitivity of the edge vector comparison,

NSIGT	an integer type in the range 3 to 7 specifying the number of standard deviations of input noise by which the edge vectors may differ and still be found acceptable.

2.3.5. Display Control

This variable controls the display panel, determining the data to be displayed and its format.

DMODE	A value from 0 to 99 for controlling the display panel. See the panel specification for a discussion of the meanings ascribed to the values.

2.4. Output Variables

The output variables of the procedure represent the results of the computations performed. They represent four classes of output: calibration, failed sensor detection, vehicle state estimation, and display panel.

2.4.1. Calibration Outputs

As noted above, the offset values for each linear accelerometer are computed from the zero acceleration calibration data. This provides an additional check on sensor functionality as well, since the standard deviation of the sensor noise can be estimated and compared with the value given. As noted above, during the discussion of the offset determination, the ratio of observed sensor noise to specified sensor noise is used to identify failed sensors during calibration at the start of a flight.

LINOFFSET(i)	gives the offset for linear accelerometer i (i = 1 to 8). Units are meters sec^{-2}.
LINNOISE(i)	is **TRUE** if sensor i (i=1 to 8) had excessive noise by the criteria given above.

If LINNOISE(i) is **TRUE**, sensor i should be marked as failed prior to the performance of any additional failed sensor tests. The values of LINNOISE and LINOFFSET must be passed to the voting routine, VOTELINOFFSET, immediately after they have been calculated and before they are used in any subsequent computations. The voting routine may or may not change their values. In any event, subsequent computations must be performed with the values returned by the voter.

2.4.2. Failed Sensor Outputs

The sensor failure detection and isolation algorithm specified below produces a vector of sensor failure data based on the inputs read during flight (RAWLIN). This vector is similar to the failed sensor input vector discussed above. Because of the

nature of the failure detection process, at most one additional failure (beyond those given in the input vector or detected during calibration) will be reported by the failure detection and isolation routine.

SYSSTATUS	is **TRUE** if at least two faces are completely operational and their edge vector satisfies the threshold test. It is **FALSE** otherwise. If the value of this variable is **FALSE**, set all elements of LINFAILOUT to **TRUE**, all channel configurations to nonoperational (0) and all acceleration estimate status indicators to UNDEFINED. The individual accelerations in LINOUT should be reported as usual.
LINFAILOUT(i)	This is the failure vector for failed sensor data outputs. It is an eight element Boolean vector with the interpretation LINFAILOUT(i) = **TRUE**, (i = 1 to 8), which implies that either LINFAILIN(i) was **TRUE** or LINNOISE(i) was **TRUE** or sensor i was determined to fail on this invocation of the procedure. At most, one entry in the vector will be found to be **TRUE** by the failure detection algorithm for linear accelerometers. Zero or more sensors may have been found to fail by the noise calibration criteria given above.

As soon as these variables have been computed, and before they are used as a basis for any vehicle state computations, it should be passed to the voter routine VOTELINFAIL. This routine may change the value you computed. If it does, computations should proceed with the changed value.

2.4.3. Vehicle State Estimation

2.4.3.1. Linear Acceleration Outputs

The redundancy management software produces outputs for each sensor in the coordinate system appropriate to the face on which the sensor is located. These values are output as well as used for vehicle state estimation.

LINOUT(i)	a real valued array representing linear acceleration component of sensor i (i=1 to 8) in the idealized Sensor Frame of Reference appropriate to the sensor. Values for failed sensors should be set to zero.

2.4.3.2. Unsolvable System

It is possible for failures of sensors to accumulate to the point at which no further failure detection is possible. The status out variables allow the procedure to specify the operational status of the system. These variables are of an enumerated type with values as follows:

NORMAL	At least 4 instrument values are available from which to estimate the 3 components of acceleration in the Navigation Frame of Reference.
ANALYTIC	Exactly 3 instrument values are available from which to compute the 3 components of acceleration in the Navigation Frame of Reference.
UNDEFINED	Fewer than 3 instrument values are available and no acceleration estimates are made.

2.4.3.3. Vehicle State Outputs

For operational purposes, five *estimates of the vehicle state* are produced. These are all estimates of acceleration in the Navigation Frame of Reference, but differ in the groupings of sensors used. Each estimate is a record which consists of a status indicator and a vector of acceleration components in the Navigation Frame of Reference. The status component takes on one of the three values described above. If the status component has the value UNDEFINED, the acceleration values should be set to zero, otherwise they should be estimated or computed as appropriate. The vehicle state output variables are:

BESTEST	The vehicle state estimated using all operational sensors, regardless of face.
CHANEST(i)	The vehicle state estimates for channel i (i=1 to 4). Note that the assignment of faces to channels varies with the health of the sensors.
CHANFACE(i)	An indication of the face pair used to compute CHANEST(i), (i = 1 to 4).

As soon as these values have been computed, and before they are used for any subsequent computations, they should be passed to the voter routine VOTEESTIMATES. This routine may or may not change the values of the variables. If it does, the changed values should be used in subsequent computations.

2.4.4. Output Panel Display

Part of the inertial unit is a display panel. The panel contains a mode indicator consisting of two seven-segment digits and two readout displays each with five seven-segment digits and a number of indicator lights for sign and decimal points. Figure 2 shows the display panel with the mode indicator at the top, and the readouts, labeled "Upper Display" and "Lower Display", below.

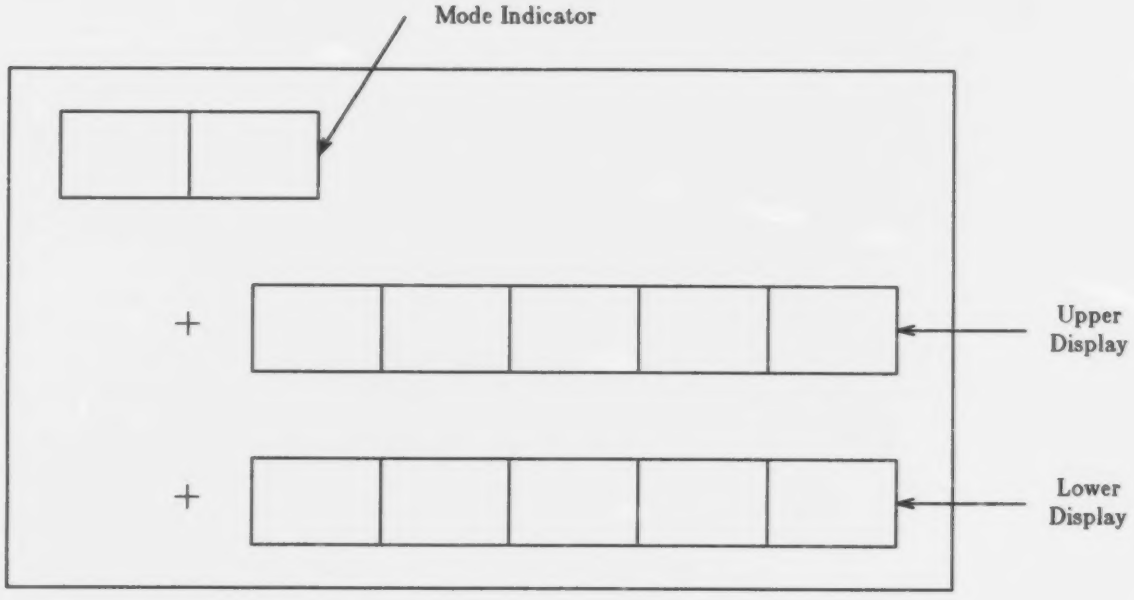


Figure 2. Display Panel

Figure 3 shows a readout display. It consists of two sign indicators, six decimal point indicators and five seven-segment digits as follows:

Component	Function
S_1	Vertical Bar.
S_2	Horizontal Bar. S_2 is used as a minus sign for displaying negative numbers. S_1 and S_2 together form a plus sign for positive numbers.
P_1	Decimal Points. These allow a display range of .00001 up to 99999.
P_2	
P_3	
P_4	
P_5	
P_6	
D_1	Digits of the display. D_1 is the most significant digit, D_5 is the least.
D_2	
D_3	
D_4	
D_5	

Figure 4 shows a typical digit. The segments are denoted "A"-"F" in a clockwise direction starting at the top with the center bar being segment "G". The mapping from segments to symbols is as follows:

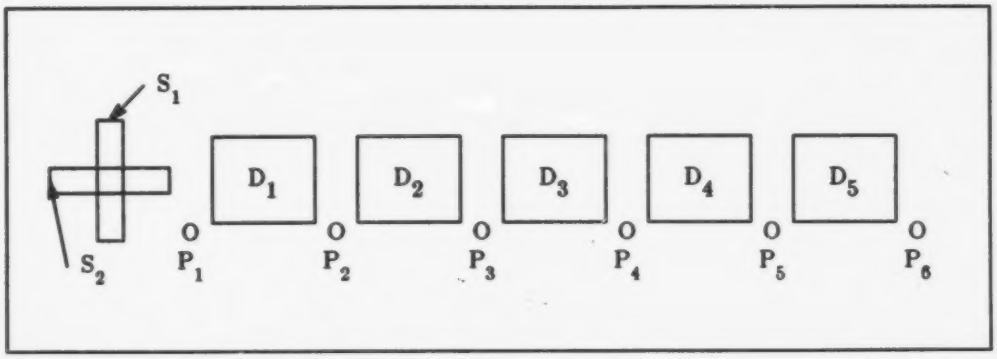


Figure 3. Typical Display

Symbol	Segments
0	ABCDEF
1	BC
2	ABDEG
3	ABCDG
4	BCFG
5	ACDFG
6	ACDEFG
7	ABC
8	ABCDEFG
9	ABCFG
A	ABCEFG
B	CDEFG
C	ADEF
D	ABCDG
F	AEFG
H	BCEFG
I	EF
N	ABEF
P	ABEFG
Blank	none

Each digit maps to seven bits, one for each segment of the display, as follows:

	MSB						LSB
Segment:	A	B	C	D	E	F	G
Bit:	6	5	4	3	2	1	0

A segment of a digit is turned on by setting its corresponding bit in a seven-bit field of an output word to a zero (the digits use negative logic). The indicator bars used for sign indication (S_1 and S_2) are also turned on by setting a zero in the appropriate bit of Word 3 of the display control variable. The decimal point indicators P_1 through P_6 use positive logic and are turned on by setting a one in the appropriate bit of Word 3

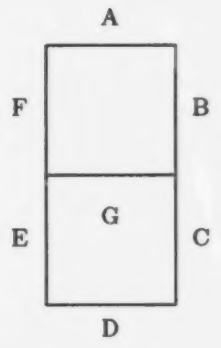


Figure 4. 7-Segment Digit

of the display control variable.

Each display is driven by a three word output as follows:

Word Descriptions

Word 1

MSB															LSB
X	X	(			D_1			)	(			D_2			)
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Word 2

MSB															LSB
X	X	(			D_3			)	(			D_4			)
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Word 3

MSB															LSB
X	S_1	S_2	P_1	P_2	P_3	P_4	P_5	P_6	(			D_5			)
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

The mode display is driven by a single word having the same format as Word 1 above. In the notation above, an X in a particular bit position indicates that that bit may be set to zero or one. (and) mark the bounds of a digit field; e.g. digit D_3 has bit 13 of Word 2 as its most significant bit and bit 7 of Word 2 as its least significant bit. The sign indicators and decimal points are assigned to individual bits of Word 3;

e.g. P_3 is assigned to bit 10 of Word 3.

The mode display has a single format consisting of unsigned positive numbers from 00 to 99 and a special case of (Blank, Blank). The upper and lower displays are capable of a number of formats. These are:

Test:	All segments and sign indicators and decimal points ON
Blank:	All segments and indicators OFF
Signed Decimal:	Signed fixed point numbers ranging from -99999. to -.00001, .00000, and +.00001 to +99999. Note that the zero value is unsigned. The value being displayed is to be rounded to 5 significant digits during conversion.
Hexidecimal:	D_1 displays the letter H. D_2-D_5 displays the hexidecimal value of a 16 bit word (i.e. 0000 through FFFF).
Failure:	D_1 shows a face indicator A, B, C, or D. D_2 and D_4 are blank. D_3 shows status for the face X axis sensor. D_5 shows status for the face Y axis sensor.

In failure mode, the following symbols will be displayed in D_3 and D_5 with the associated meanings implied:

Display	Meaning
P	Sensor is operational.
N	Sensor marked as failed due to excessive noise during calibration.
I	Sensor was marked as failed on input.
F	Sensor failure detected by redundancy management algorithm.

The display panel can be used to display a variety of raw and derived quantities as well as the sensor status. The type of display is determined by the input variable DMODE. The following specification describes the modes of interest:

Mode: 00,05-20,25-30,34-87,89-99

Upper Display: Blank

Lower Display: Blank

Modes: 01-04

Upper Display: Blank

Lower Display: Failure format for faces A-D respectively

Modes: 21-24

Upper Display: Linear acceleration raw data for axes X_{IA}-X_{ID} respectively. This is bits 11-0 of the sensor input word displayed in hexidecimal format.

Lower Display: As above for axes Y_{IA} -Y_{ID} respectively.

Modes: 31-33

Upper Display: Linear acceleration along the X_N-Z_N axes respectively in signed decimal format.

Lower Display: Blank

Mode: 88

Upper Display: Test Format

Lower Display: Test Format

The display panel outputs are contained in the following variables:

DISMODE	Output for the two digit mode display in "Word 1" format.
DISUPPER(i) DISLOWER(i)	Three word arrays of 16 bit integers to drive the upper and lower displays. The first element has "Word 1" format, the second "Word 2" format, the third "Word 3" format.

The computed values of the display control words must be passed to the voting routine VOTEDISPLAY as soon as they have been calculated. Since these constitute the final output of your program, they are not used in subsequent computations, however if the voter alters their values, the altered values must be left in the variables when your procedure terminates.

2.5. Sensor Failure Detection And Isolation

You are to use the edge vector test method described in this section to detect and isolate accelerometer failures. The *edge vector test* is based on resolving the sensor outputs along the edges of the semioctahedron for comparison. In this section, it is assumed that the raw accelerometer measurements have been converted into appropriate engineering units, and compensated for misalignment so that, for each triangular face of the semioctahedron, there are two accelerometer measurements expressed along

the two appropriate axes of the idealized Sensor Frame of Reference. The projections of the sensor frame axes perpendicular to the faces are to be ignored. A mathematical specification of the edge vector test is given in Appendix D.

The two linear accelerometers in each face of the semioctahedron are sufficient to compute the components of the vehicle acceleration which lie along arbitrary lines within the face.

Each face of the semioctahedron lies in a plane which has a line of intersection with the plane containing each other face of the semioctahedron. For adjacent faces, these lines contain the edges of the semioctahedron. For opposite faces, the line is parallel to the bases of the two faces in question and passes through the apex of the semioctahedron.

If the four accelerometer sensors in two faces are operational, the component of each face acceleration along the line of their intersection should be the same within the tolerances determined by the noise present in the sensors. These tolerance thresholds are specified in Appendix D. Under the assumption that at most a single additional sensor will fail for a given execution, the face containing the failed sensor can be identified by comparing the faces pairwise and eliminating the face common to all out of tolerance comparisons.

The two linear accelerometers of a face are independent and fail separately. When a bad face is detected, the determination of the failed accelerometer sensor axis is to be done as follows. Compute the least squares estimate of the specific force on the RSDIMU in the Instrument Frame of Reference, as described in Appendix C,

using only those accelerometers on faces determined to be good by the edge vector test. Using this computed specific force, estimate the specific force along the axis of each sensor in the questionable face by finding the projection of the computed specific force on these sensor axes. The failed accelerometer is identified by taking the difference between the specific force estimate and the actual measurement. If the absolute value of this difference is greater than NSIGT x SIGMA for a given axis, then that accelerometer is declared to be failed. NSIGT is an integer from {3,4,5,6,7} and SIGMA is the average of LINSTD converted to program engineering units, using the calibration for the sensors involved.

If a specific accelerometer on a given face is declared to be failed prior to the invocation of the program, then that face is not used in the edge vector test. However, the health of the functioning axis on that face is determined according to the same procedure outlined above.

If a single sensor on a face fails, the face cannot be used in the edge vector test. When two faces have failures, only a single edge is available and further sensor isolation is not possible with this method. Should the system be in a configuration where only a single pair of faces is available for the edge vector test, i.e., only two faces have both sensors operational and the edge vectors fail the threshold test, the system must be declared nonoperational and no acceleration estimates can be made. This will be the case even though it appears that enough accelerometers are functioning to make an estimate since it is no longer possible to determine which accelerometers are functioning and which have failed by the edge vector method.

Outputs of this computation should be reported in the output variables SYSSTATUS, LINFAILOUT, and LINOUT.

2.6. Vehicle Acceleration

Given the set of operational sensors determined by the sensor feature detection and isolation test, either an overdefined, exactly defined, or underdefined system of equations will exist. If either of the first two cases occurs, compute the least squares estimate as described in Appendix C or analytical solution of the vehicle linear acceleration in the Navigation Frame of Reference as appropriate using all functional accelerometers. This is output in the variable BESTEST.

In addition, your procedure is to provide, for each of the four channels described above, a best estimate of the inertial acceleration based on the measurements solely from sensors on the faces associated with that single channel. These will be referred to as *channel estimates* and will be reported in the variable CHANEST. Note that the mapping from face pairs to channels changes as sensors fail. For a channel to function, at least three of the four sensors associated with the channel must function properly. Thus, a failure of one sensor on face A with all other sensors operational would result in channels 1 and 4 having ANALYTIC status while channels 2 and 3 would retain NORMAL status. The additional failure of a sensor on face B would result in channel 1 becoming UNDEFINED, channels 2 and 4 becoming ANALYTIC and channel 3 retaining NORMAL status. If on the other hand, the second sensor on face A failed, the system would be reconfigured so that channel 1 would be nonoperational, channels 2, 3, and 4 would have NORMAL status, but channel 4 would now consist of faces D

and B rather than A and D. The face pair associated with each channel is reported in the variable CHANEST using the mapping described above.

Note that if one accelerometer on a given face is determined to be failed while the other is nct, then the functioning accelerometer on that face is used to compute channel estimates involving the face in question. In these computations, the measurement compensated for misalignment is not to be used since the compensated measurement would be corrupted during the occurrence of a fault in the other axis. On subsequent entries into the program, the misalignment compensation cannot be performed with one functioning accelerometer. Hence, in this case, the channel estimates are to be computed using the measurement not compensated for misalignment on the face with one functioning accelerometer.

3. PASCAL DECLARATIONS

The following listings contain the contents of four files which you are to include in your program. These files define constants, types, and variables which provide the interface between your program and its external environment. Also included are a set of voting routines which permit your procedure to be used as part of a fault tolerant system. The voting routines allow intermediate results of your computations to be compared with other versions of the program operating on other processors.

3.1. Constants

```
{ The following constants are defined for your use in the
  program. }

const

  ALB = 1;
  AUB = 3; { Bounds for axes of Frames of Reference arrays}

  CHLB = 1;
  CHUB = 4; { Bounds for channel array }

  CLB = 1;
  CUB = 50; { Bounds for calibration array }

  DLB = 1;
  DUB = 3; { Bounds for display register arrays }

  ELB = 1;
  EUB = 6; { Bounds for misalignment (error) angle arrays }

  FLB = 1;
  FUB = 4; { Bounds for face-oriented arrays }

  G = 32.0; { Gravitational constant in ft/sec2}

  MAXINT = 65535;

  MODEMIN = 0;
```

```
MODEMAX = 99; { Range values for display modes }

MLB = 0;
MUB =  MAXINT; { Bounds for nonnegative, unsigned 16 bit
                 machine integer }

NSIGMIN = 3;
NSIGMAX = 7; { Bounds for NSIGT }

PI = 3.1415926535;

PAIRMIN = 0;
PAIRMAX = 6; { Bounds for face-pair arrays }

SLB = 1;
SUB = 8; { Bounds for sensor and related arrays }
```

3.2. Types

```
{ The following types are defined for use in your program.
  To avoid portability problems during program evaluation,
  ALL variables which are based on integer or real types
  must be declared using one of these types or a subtype
  formed from one of them. }

type
  { Base types for portability }
  IINT = integer;
  IREAL = real;

  { Index types }
  AINDEX = ALB..AUB;
  CHINDEX = CHLB..CHUB;
  CINDEX = CLB..CUB;
  DINDEX = DLB..DUB;
  EINDEX = ELB..EUB;
  FINDEX = FLB..FUB;
  MODET = MODEMIN..MODEMAX;
  NSIGSET = NSIGMIN..NSIGMAX;
  PAIRT = PAIRMIN..PAIRMAX;
  SINDEX = SLB..SUB;
```

```
MINT = MLB..MUB;
       { represents a 16-bit nonnegative machine integer }

SYSTEM = (NORMAL,ANALYTIC,UNDEFINED);
         { Computational modes }

ARARRAY  = array [AINDEX] of IREAL;
           { holds information for 3 axes }

CMARRAY = array [SINDEX, CINDEX] of MINT;
          { holds calibration data points for
            eight accelerometers }

CPARRAY = array [CHINDEX] of PAIRT;
          { holds facepair used to compute channel
            estimate }

DMARRAY = array [DINDEX] of MINT;
          { holds 'words' of display information }

ERARRAY  = array [FINDEX, EINDEX] of IREAL;
           { holds misalignment angles of eight
             accelerometers }

FRARRAY = array [FINDEX] of IREAL;

SBARRAY = array [SINDEX] of boolean;
          { holds sensor failure indications }

SIARRAY = array [SINDEX] of IINT;
          { holds raw data for 8 accelerometers }

SMARRAY = array [SINDEX] of MINT;
          { holds count data for 8 accelerometers}

SRARRAY = array [SINDEX] of IREAL;
          { holds slope coefficients for 8 accel's }

STATE = record
          status: SYSTEM;
          acceleration: ARARRAY
        end;  { Holds one vehicle state estimation }

VSEARRAY= array [CHINDEX] of STATE;
```

```
                { holds vehicle state estimation of 4 channels }
```

3.3. Variables

```
{ The following variables are global to the routine you
  are writing.  Input variables have well defined values
  upon entry to your code.  Output variables should be
  considered as having no defined values. }

var

  { Input Variables }

  OBASE : IREAL; { Semi-octahedron base }

  OFFRAW : CMARRAY;
           { Calibration data for 8 accelerometers }

  LINSTD : MINT;
           { Noise standard deviation (in counts)
             for accelerometers }
  LINFAILIN : SBARRAY;
              { Accelerometer failure initial conditions }

  RAWLIN : SMARRAY;
           { Raw data for acceleration computation }

  DMODE : MODET; { Display mode }

  TEMP : FRARRAY; { current temperature on each face }

  SCALE0, SCALE1, SCALE2 : SRARRAY;
           { Linear accelerometer slope coefficients }

  MISALIGN : ERARRAY; { Accelerometer misalignment angles }

  NORMFACE : FRARRAY; { Accelerations normal to I faces }

  NSIGT : NSIGSET; { Noise tolerance }

  PHIV, THETAV, PSIV : IREAL; { N to V Rotations }
  PHII, THETAI, PSII : IREAL; { V to I Rotations }
```

```
{ Output Variables }

  LINOFFSET : SRARRAY;
  LINNOISE : SBARRAY; { Sensor calibration results }

  LINFAILOUT : SBARRAY; { Failure detection results }

  LINOUT : SRARRAY; { Individual sensor outputs }

  DISMODE : MINT;  { Display panel mode }

  DISUPPER : DMARRAY;
  DISLOWER : DMARRAY; { Display panel encodings }

  BESTEST : STATE;
            { Vehicle State Estimate using all
              operational sensors}

  CHANEST : VSEARRAY;
            { Vehicle State Estimates for the four channels }

  CHANFACE : CPARRAY; { Maps face pairs to channels }

  SYSSTATUS : boolean; { Operational status of system }
```

3.4. Voters

```
{ These routines are stubs for voting routines containing
  calls to a distributed system voter.  Prior to acceptance
  testing, bodies will be supplied for the routines. }

procedure VOTELINOFFSET (var LINOFFSET : SRARRAY;
                    { Sensor offsets }
                         var LINNOISE : SBARRAY
                             { Sensor noise test results });

procedure VOTELINFAIL (var SYSSTATUS : boolean;
                           { Operational status of system }
                       var LINFAILOUT : SBARRAY
                           { Failure detection results });


procedure VOTEESTIMATES (var BESTEST : STATE;
```

```
                                { Vehicle State Estimate using
                                  all operational sensors}
                            var CHANEST : VSEARRAY;
                                { Vehicle State Estimates
                                  for the four channels }
                            var CHANFACE : CPARRAY
                                { Maps face pairs to channels });

procedure VOTEDISPLAY (var DISMODE : MINT;
                           { Mode Display }
                       var DISUPPER : DMARRAY;
                           { Upper Display }
                       var DISLOWER : DMARRAY
                           { Lower Display });
```

4. BIBLIOGRAPHY

1. "Preliminary Design of an RSDIMU Using Two-Degree-of-Freedom Tuned-Gimbal Gyroscopes," NASA CR-145035 (October 1976).
2. F.R. Morrell and J.G. Russell, "Design of a Developmental Dual-Fail Operational RSDIMU," *Proceedings of IEEE NAECON*, (May 1980).
3. P. Motyka, M. Lantey, and R. McKern, "Failure Detection and Isolation Analysis of a RSDIMU," NASA CR-165658 (February 1981).
4. F.R. Morrell and P. Motyka, "Unified Analysis Methods for a Fault Tolerant RSDIMU," *Proceedings of the AIAA*, (November 1983). 5th Digital Avionics Systems Conference
5. W.H. Bryant, F.R. Morrell, and M.L. Bailey, "Flight Test Configuration for Verifying Inertial Sensor Redundancy Management Techniques," *Proceedings of the AIAA*, (November 1984). Aircraft Design, Systems, and Operations Meeting
6. K.R. Britting, *Inertial Navigation Systems Analysis,* John Noley & Sons, Inc., New York, New York (1971).
7. P. Lancaster, *Theory of Matrices,* Academic Press, New York (1969).
8. E.A. Mechtly, "The International System of Units," NASA-SP-7012 (1973).

5. APPENDIX A - PROBLEM COORDINATE FRAMES

We specify here a number of coordinate systems (or "frames") necessary to the definition of the problem. All of the systems are right-handed, and all but the last described here are orthogonal.

5.1. Local Navigation Frame (N)

The origin of this frame is some point on the earth, "near" the vehicle. The x_N, y_N, and z_N axes of this frame are aligned along the north, east, and down directions, as sketched in figure A1.

Also shown is the local gravity vector **g**, specifying the acceleration due to gravity; note that it is aligned with z_N.

For simplicity, we assume the earth is non-rotating (and non-translating), so that we can regard the N frame as an "inertial" frame of reference; i.e., one in which

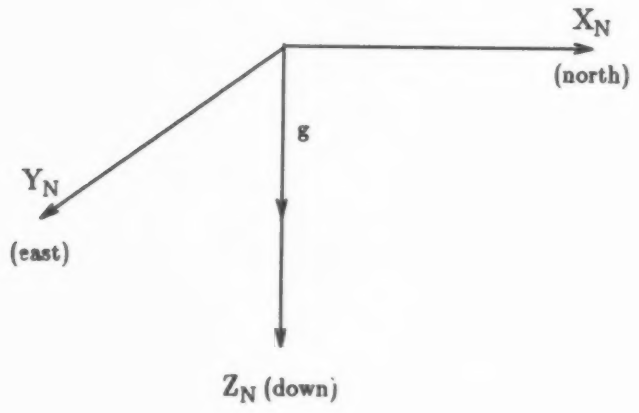


Figure A1. Local Navigation Frame

Newton's Laws hold.

5.2. Vehicle Frame (V)

The origin of this frame is at the vehicle center-of-gravity (CG). The x_v, y_v, and z_v axes of this frame are aligned so as to point to the vehicle's nose, out the right wing, and down, as illustrated in figure A2.

The location of the vehicle CG, and hence the V-frame origin, with respect to the N-frame origin, is given by the three dimensional (3D) vector **r**. This is illustrated in figure A3a. The representation of the vehicle position vector, **r**, in the N-frame is defined by:

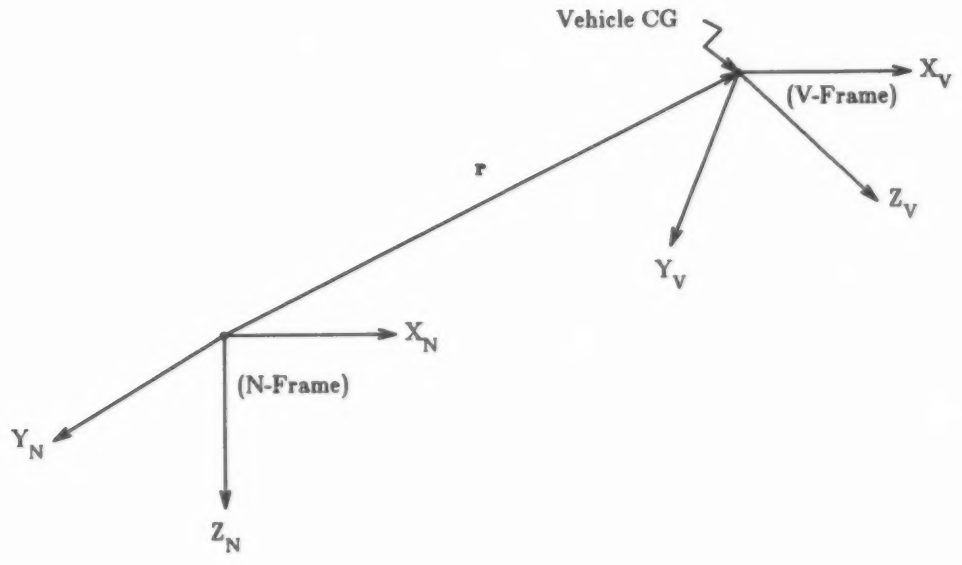


Figure A3a. Location of V-Frame w.r.t. N-Frame

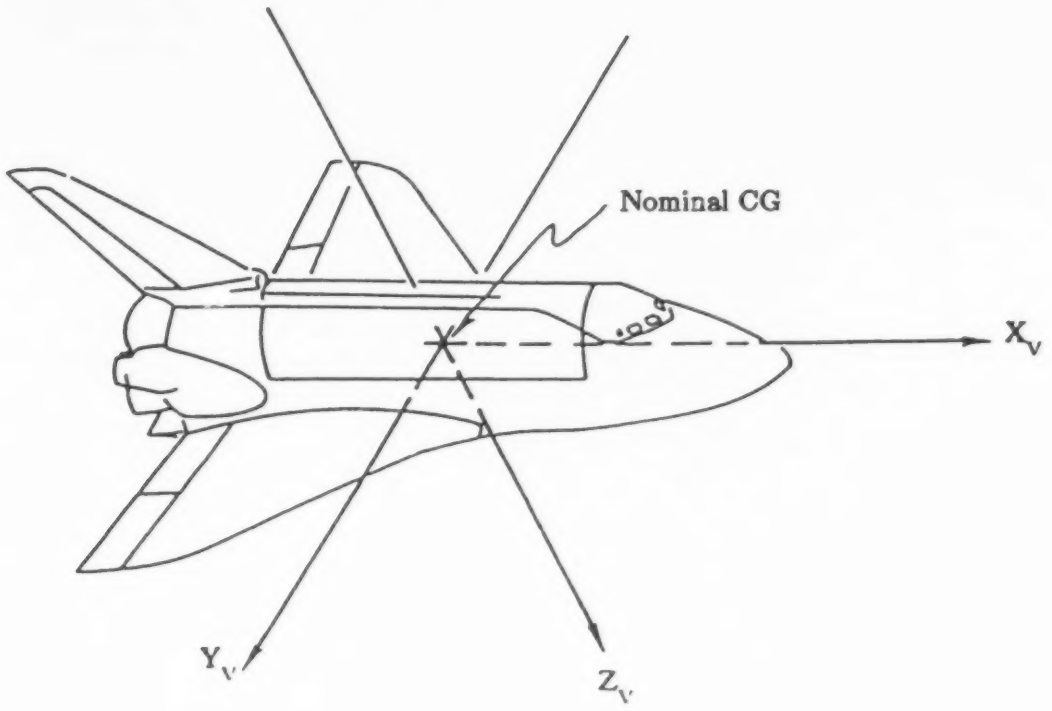


Figure A2. Relation of V-Frame to Vehicle

$$\mathbf{r}^N \equiv \begin{bmatrix} x_{NV} \\ y_{NV} \\ z_{NV} \end{bmatrix} \tag{A1}$$

where the superscript N above denotes the coordinate system in which the vector **r** is expressed. Coordinate representation by a superscript will be used throughout the appendices.

The orientation of the V-frame with respect to the N-frame is defined by an "Euler Rotation Sequence" which serves to bring the N-frame into coincidence with the V-frame. This rotation sequence is illustrated in figure A3b, for the yaw, pitch, roll sequence given by ψ_V, θ_V, and ϕ_V, respectively.

To transform vectors between the different frames shown, we note that, for arbitrary 3D vector, **a**,

$$\mathbf{a}' \equiv \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos(\psi_V) & \sin(\psi_V) & 0 \\ -\sin(\psi_V) & \cos(\psi_V) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_N \\ y_N \\ z_N \end{bmatrix} \equiv T_1(\psi_V)\mathbf{a}^N \tag{A2a}$$

where $T_1(\psi_V)$ is the transformation matrix representing the yaw rotation about the Z_N axis by an angle ψ_V;

$$\mathbf{a}'' \equiv \begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix} = \begin{bmatrix} \cos(\theta_V) & 0 & -\sin(\theta_V) \\ 0 & 1 & 0 \\ \sin(\theta_V) & 0 & \cos(\theta_V) \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \equiv T_2(\theta_V)\mathbf{a}' \tag{A2b}$$

where $T_2(\theta_V)$ is the transformation matrix representing the pitch rotation about the Y' axis by an angle θ_V;

$$\mathbf{a}^V \equiv \begin{bmatrix} x_V \\ y_V \\ z_V \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi_V) & \sin(\phi_V) \\ 0 & -\sin(\phi_V) & \cos(\phi_V) \end{bmatrix} \begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix} \equiv T_3(\phi_V)\mathbf{a}'' \tag{A2c}$$

where $T_3(\phi_V)$ is the transformation matrix representing the roll rotation about the x'' axis and by an angle ϕ_V. Note that the primes here denote rotated axes and not vector transposition. Thus, the transformation of an arbitrary 3D vector **a** from the N-frame into the V-frame is accomplished via:

$$\mathbf{a}^V = T_{VN}\ \mathbf{a}^N \tag{A3a}$$

where T_{VN} is the coordinate transformation matrix, mapping the N-frame representation of a vector into its V-frame representation, defined by the matrix product of the single axis rotation matrices given in (A2a-c):

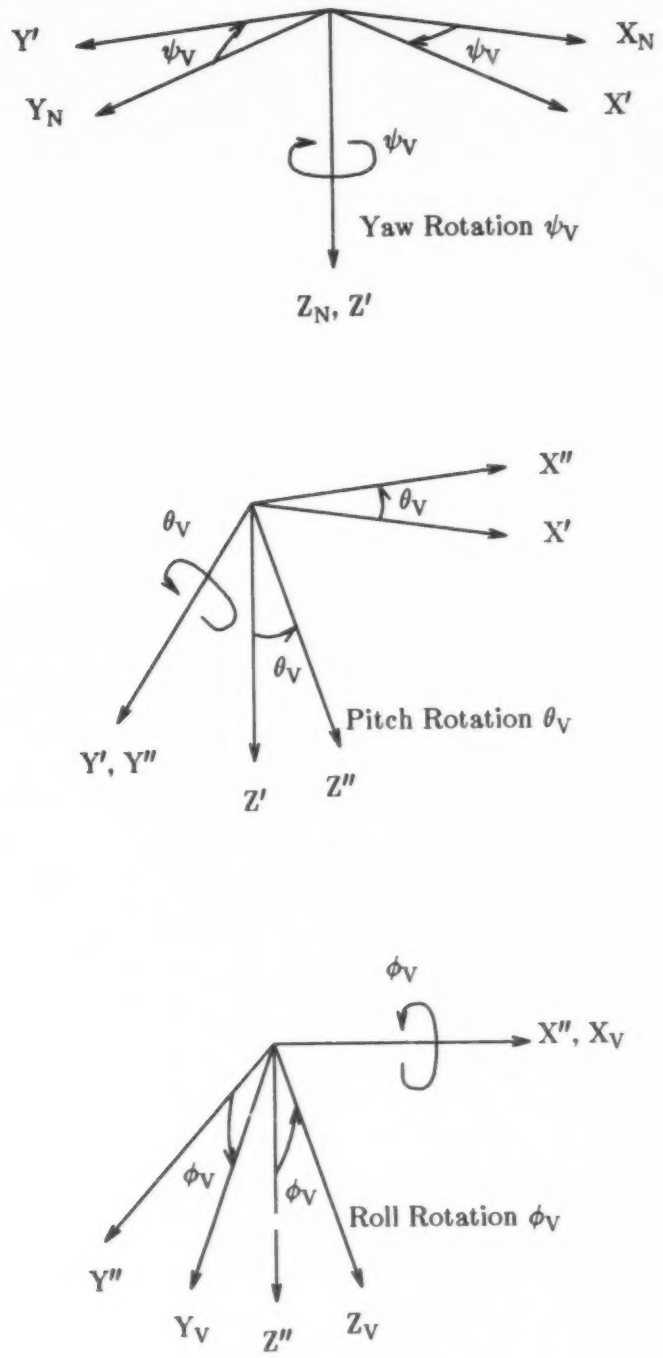


Figure A3b. Euler Sequence for N to V Frame Transformation

$$T_{VN} = T_3(\phi_V)\, T_2(\theta_V)\, T_1(\psi_V) \tag{A3b}$$

Consistent with our notation on superscripts, $\mathbf{a}^N$ denotes the representation of the **a** vector in the N-frame whereas $\mathbf{a}^V$ denotes its representation in the V-frame. The subscript notation on the coordinate transformation matrix in (A3a) will be used throughout the appendices; e.g., a transformation from the P Frame to the Q Frame is accomplished via multiplication by the transformation matrix T_{QP}.

The coordinate transformation matrix from the V-frame into the N-frame is similarly constructed from the single axis rotation matrices by reversing both the sequence and sense of the rotations. That is,

$$\mathbf{a}^N = T_{NV}\ \mathbf{a}^V \tag{A4a}$$

and

$$T_{NV} = T_1(-\psi_V)\, T_2(-\theta_V)\, T_3(-\phi_V) \tag{A4b}$$

where T_{NV} is the coordinate transformation matrix, mapping the V-frame representation of a vector into its N-frame representation, and $T_3(-\phi_V)$, $T_2(-\theta_V)$ and $T_1(-\psi_V)$ are computed in accordance with (A2) with the indicated angles.

5.3. Instrument Frame (I)

The origin of this frame is at the centroid of the base of the semioctahedron. As shown in figure A4, the axes are aligned with the semioctahedron so that the z_I axis points to the apex, and the x_I and y_I axes point to adjacent base corners.

Faces are labeled A, B, C, D, with the base edge of face A contained in the first quadrant of the $x_I - y_I$ plane, and subsequent faces B through D proceeding in a clockwise direction about the z_I axis when looking down from the vertex.

The origins of the I and V frames are co-located. The orientation of the I-frame with the respect to the V-frame is defined by an Euler Rotation Sequence, which brings the V-frame into coincidence with the I-frame. This rotation sequence is illustrated in figure A5, for the yaw, pitch, roll sequence defined by the rotation angles ψ_I, θ_I, ϕ_I, respectively.

This sequence is directly analogous to the sequence used for the N-frame to V-frame transformation. Accordingly, on the basis of (A2) - (A4), for arbitrary **a**,we have

$$\mathbf{a}^I = T_{IV}\, \mathbf{a}^V \tag{A5a}$$

where the coordinate transformation matrix T_{IV}, mapping the V-frame representation of a vector into its I-frame representation, is defined by:

$$T_{IV} \equiv T_3(\phi_I)\, T_2(\theta_I)\, T_1(\psi_I) \tag{A5b}$$

where $T_1(\psi_I)$, $T_2(\theta_I)$ and $T_3(\phi_I)$ are computed in accordance with (A2) using the angles ψ_I, θ_I, ϕ_I.

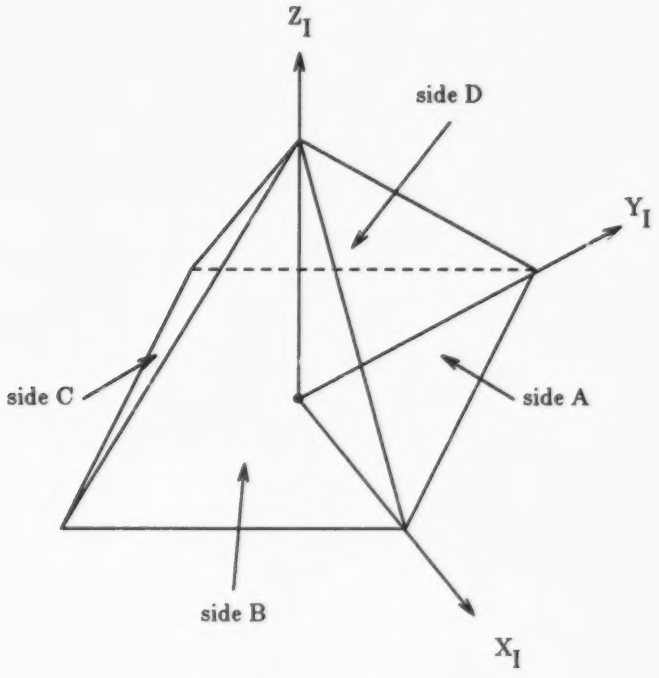
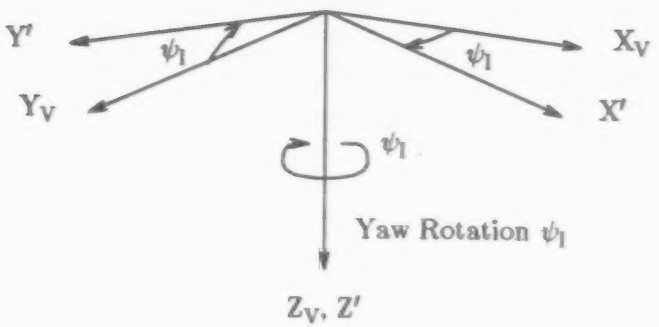
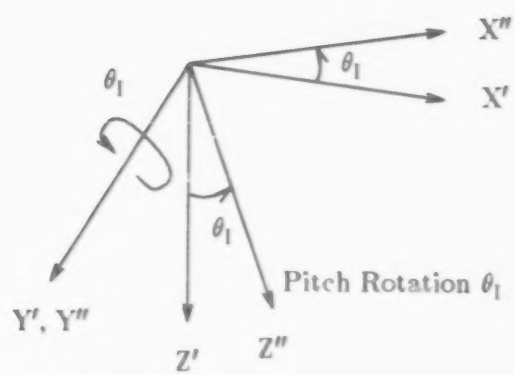


Figure A4. Relation of I-Frame to Semi-Octahedron





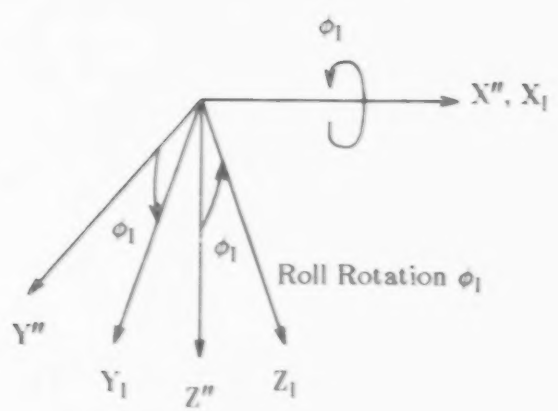


Figure A5. Euler Sequence for V to l Frame Transformation

The coordinate transformation matrix, T_{VI}, mapping the I-frame representation of a vector into its V-frame representation, can be constructed by reversing both the sequence and sense of rotations in (A5b) as done in (A4b) for the case of V-frame to N-frame transformation.

5.4. Sensor Frames (A, B, C, D)

There are four sensor frames, one associated with each triangular face of the semioctahedron. They are labeled A, B, C, D, in accordance with the face labeling introduced in figure A4 above. The A-frame is illustrated in figure A6; frames B through D are defined analogously.

As shown in the figure, the origin of the A-frame is located at the centroid of the A-face. The axes are aligned so that the z_A axis is normal to the face, pointing outward. The x_A and y_A axes are in the plane of the A-face, and are symmetrically oriented with respect to the perpendicular bisector of the base edge, both pointing downwards to that edge.

For the A-Sensor Frame, if we associate with each axis (x_A, y_A, z_A) a unit-length vector which is aligned with that axis, and which points in the positive direction for that axis, we can define three corresponding A-frame unit vectors ($\mathbf{x}_A$, $\mathbf{y}_A$, $\mathbf{z}_A$). It can be shown from the geometry of the semioctahedron that these can be expressed in I-frame coordinates as follows:

$$\mathbf{x}_A^I = \frac{1}{2\sqrt{3}}\begin{bmatrix}\sqrt{3}+1\\ -\sqrt{3}+1\\ -2\end{bmatrix};\ \mathbf{y}_A^I = \frac{1}{2\sqrt{3}}\begin{bmatrix}\sqrt{3}+1\\ -\sqrt{3}+1\\ -2\end{bmatrix};\ \mathbf{z}_A^I = \frac{1}{\sqrt{3}}\begin{bmatrix}1\\ 1\\ 1\end{bmatrix} \qquad \text{(A6a,b,c)}$$

Likewise, for the B, C, & D frames:

$$\mathbf{x}_B^I = \frac{1}{2\sqrt{3}}\begin{bmatrix}-\sqrt{3}+1\\ -\sqrt{3}-1\\ -2\end{bmatrix};\ \mathbf{y}_B^I = \frac{1}{2\sqrt{3}}\begin{bmatrix}\sqrt{3}+1\\ \sqrt{3}-1\\ -2\end{bmatrix};\ \mathbf{z}_B^I = \frac{1}{\sqrt{3}}\begin{bmatrix}1\\ -1\\ 1\end{bmatrix} \qquad \text{(A7a,b,c)}$$

$$\mathbf{x}_C^I = \frac{1}{2\sqrt{3}}\begin{bmatrix}-\sqrt{3}-1\\ \sqrt{3}-1\\ -2\end{bmatrix};\ \mathbf{y}_C^I = \frac{1}{2\sqrt{3}}\begin{bmatrix}\sqrt{3}-1\\ -\sqrt{3}-1\\ -2\end{bmatrix};\ \mathbf{z}_C^I = \frac{1}{\sqrt{3}}\begin{bmatrix}-1\\ -1\\ 1\end{bmatrix} \qquad \text{(A8a,b,c)}$$

$$\mathbf{x}_D^I = \frac{1}{2\sqrt{3}}\begin{bmatrix}\sqrt{3}-1\\ \sqrt{3}+1\\ -2\end{bmatrix};\ \mathbf{y}_D^I = \frac{1}{2\sqrt{3}}\begin{bmatrix}-\sqrt{3}-1\\ -\sqrt{3}+1\\ -2\end{bmatrix};\ \mathbf{z}_D^I = \frac{1}{\sqrt{3}}\begin{bmatrix}-1\\ 1\\ 1\end{bmatrix} \qquad \text{(A9a,b,c)}$$

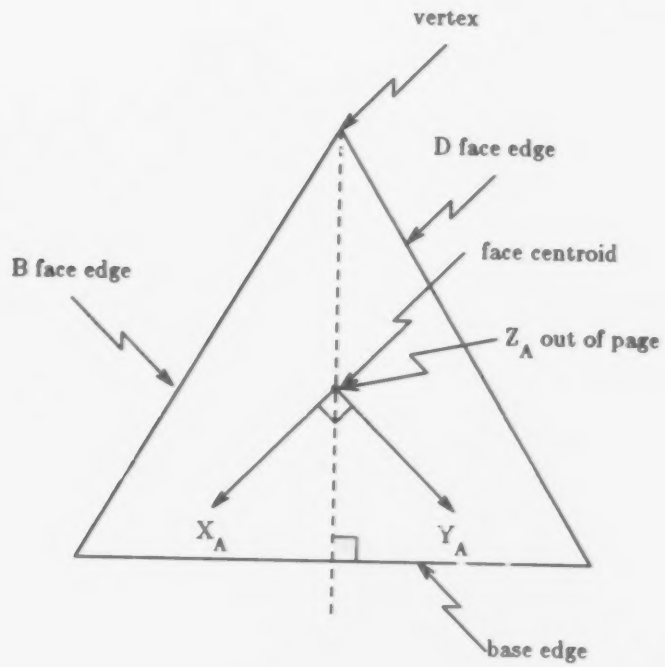


Figure A6. Relation of A-Frame to A-Face of Semi-Octohedron

Transformation of an arbitrary 3D vector, **a**, from the I-frame into Sensor Frame for face A is accomplished via:

$$\mathbf{a}^A = T_{AI}\mathbf{a}^I \tag{A10a}$$

where $\mathbf{a}^I$ is the representation of the vector **a** in the I-frame, $\mathbf{a}^A$ is the representation of the vector **a** in the A-Sensor Frame, and T_{AI} is the coordinate transformation matrix, mapping the I-frame representation of a vector into its A-Sensor Frame representation, defined by:

$$T_{AI} \equiv \begin{bmatrix} \mathbf{x}'_A \\ \mathbf{y}'_A \\ \mathbf{z}'_A \end{bmatrix} \tag{A10b}$$

where the $'$ superscript denotes a transpose, and where $\mathbf{x}_A$, $\mathbf{y}_A$, and $\mathbf{z}_A$ are given by (A6a-c). Transformation of an arbitrary 3D vector $\mathbf{a}$ from the Sensor Frame A into the I-frame is accomplished via:

$$\mathbf{a}^I = T_{IA}\mathbf{a}^A \tag{A11a}$$

where T_{IA} is the 3x3 coordinate transformation matrix, mapping the A-Sensor Frame coordinates into I-Frame coordinates, defined by:

$$T_{IA} = [\mathbf{x}_A, \mathbf{y}_A, \mathbf{z}_A] \tag{A11b}$$

where $\mathbf{x}_A$, $\mathbf{y}_A$, and $\mathbf{z}_A$ are given by (A6a-c). The transformation matrices, mapping I-frame coordinates into B, C, and D Sensor Frame coordinates, are similarly defined.

We have specified the orientation of each Sensor Frame (A, B, C, D). To specify the location of the origin of each frame, note that the distance from the base face centroid (i.e., the origin of the I-frame), to the centroids of any of the four triangular faces (i.e., the origins of the A, B, C, and D frames) is given by $\frac{d}{\sqrt{6}}$, where d is the length of an edge of the semioctahedron.

The location of the origin of Sensor Frame A, with respect to the I-frame origin, is then given by the 3D vector $\mathbf{r}_{IA}$, where

$$\mathbf{r}_{IA} = \frac{d}{\sqrt{6}}\mathbf{z}_A \tag{A12}$$

where $\mathbf{z}_A$ is the outward pointing unit-length vector normal to face A. Substitution of (A6c) into above specifies this normal vector in the I-frame. The normal vectors representing the locations of the B, C, and D Sensor Frame origins in the I-frame are similarly defined.

5.5. Measurement Frames ($\bar{A}$, $\bar{B}$, $\bar{C}$, $\bar{D}$)

There are 4 measurement frames ($\bar{A}$, $\bar{B}$, $\bar{C}$, $\bar{D}$), one associated with each sensor frame {A, B, C, D}. The origins of sensor and measurement frames for a given face are co-located.

Each axis of a measurement frame on a given face is "misaligned" by some "small" amount from the corresponding axis of the sensor frame on that face. We illustrate this in figure A7, for a generic sensor-frame x axis, which is misaligned by the two angles θ_{xy} and θ_{xz}, to yield the generic measurement-frame $\bar{x}$ axis. The first subscript denotes the axis being misaligned (x), while the second specifies the axis about which the misalignment rotations are measured (y or z). Note that the angles are

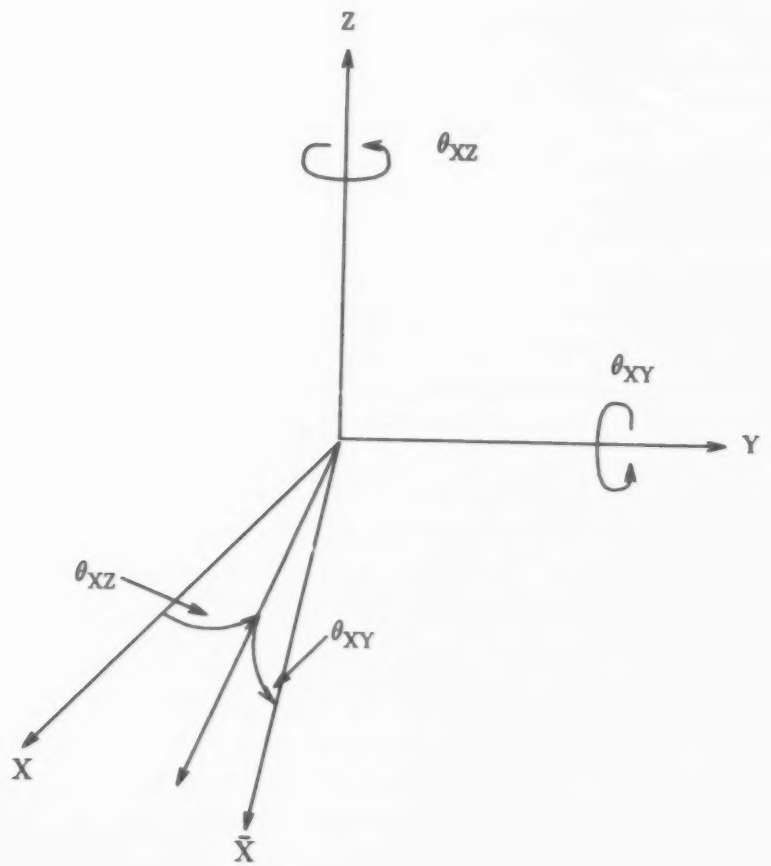


Figure A7. Misalignment Angles for X-Axis

"small" (i.e., $<<$ 1 when measured in radians), so that rotation order is unimportant. Note also that positive misalignments are defined in accordance with the right-hand rule.

The misalignment for the y- and z-axes are specified in a similar fashion. What results is a measurement frame defined by the $\bar{x}$, $\bar{y}$, $\bar{z}$ axes, which, in general will be nonorthogonal.

The coordinate transformation of an arbitrary 3D vector **a** from the orthogonal Sensor Frame A into the misaligned Measurement Frame $\bar{A}$ is accomplished via:

$$\bar{\mathbf{a}}^{A} = T_{\bar{A}A}\, \mathbf{a}^{A} \tag{A13a}$$

where $\mathbf{a}^{A}$ is the representation of the vector **a** in the Sensor Frame A, and $\bar{\mathbf{a}}^{A}$ is the representation of the vector **a** in the Measurement Frame $\bar{A}$, and $T_{\bar{A}A}$ is the transformation matrix, mapping the A-Sensor Frame coordinates into the $\bar{A}$-Measurement Frame coordinates, defined by:

$$T_{\bar{A}A} \equiv \begin{bmatrix} 1 & \theta^{A}_{xz} & -\theta^{A}_{xy} \\ -\theta^{A}_{yz} & 1 & \theta^{A}_{yx} \\ \theta^{A}_{zy} & -\theta^{A}_{zx} & 1 \end{bmatrix} \tag{A13b}$$

The transformation of an arbitrary 3D vector **a** from the misaligned Measurement Frame $\bar{A}$ to the orthogonal Sensor Frame A is accomplished via:

$$\mathbf{a}^{A} = T_{A\bar{A}}\bar{\mathbf{a}}^{A} \tag{A14a}$$

where $T_{A\bar{A}}$ is the transformation matrix, mapping the $\bar{A}$ - Measurement Frame coordinates into A-Sensor Frame coordinates, defined by:

$$T_{A\bar{A}} \equiv \begin{bmatrix} 1 & -\theta^{A}_{xz} & \theta^{A}_{xy} \\ \theta^{A}_{yz} & 1 & -\theta^{A}_{yx} \\ -\theta^{A}_{zy} & \theta^{A}_{zx} & 1 \end{bmatrix} \tag{A14b}$$

The transformation matrices mapping the A, B, C, and D Sensor Frame coordinates into the corresponding $\bar{B}$, $\bar{C}$, and $\bar{D}$ Measurement Frame coordinates are similarly defined. Hence, in general, we will have 24 misalignment angles specified (six per face).

6. APPENDIX B - ACCELEROMETER MEASUREMENT EQUATIONS

In this section, we give a mathematical description for the accelerometer measurements by specifying the functional relationships between the true accelerometer outputs and the inertial vehicle acceleration. By a "true" accelerometer, we mean a sensor without any bias, scale factor, and noise errors. Moreover, as discussed in Appendix A, the inertial vehicle acceleration, which your program is to compute, is the vehicle acceleration as seen by an observer fixed with respect to the Navigation (N) Frame of Reference.

An accelerometer does not measure the inertial vehicle acceleration directly but rather measures the specific force exerted on the RSDIMU. Specific force, which is the difference between the inertial acceleration and acceleration due to gravity, is specified below.

6.1. Specific Force

For the purpose of this application, we assume that the accelerometers are mounted at the face centroids of the semioctahedron. Moreover, we assume that the lever arm effects due to the separation between the face centroids and the Instrument (I) Frame origin can be neglected.

Under these assumptions, referring to Figure B1, the true specific force vector, $\mathbf{f}$, is defined by, and its representation in the Navigation and Sensor Frame of Reference on face A, are given by:

$$\mathbf{f} \equiv \mathbf{a} - \mathbf{g} \tag{B1a}$$

$$\mathbf{f}^N = \mathbf{a}^N - \mathbf{g}^N \tag{B1b}$$

$$\mathbf{f}^A = T_{AN}\ \mathbf{f}^N \tag{B1c}$$

where $\mathbf{a}^N$ is the inertial vehicle acceleration to be computed by your program, which is the second time derivative of the vehicle position vector, $\mathbf{r}$, expressed in the N-Frame. The 3x3 matrix T_{AN} is the coordinate transformation matrix from the N-Frame to A-Sensor Frame defined by:

$$T_{AN} = T_{AI}\ T_{IV}\ T_{VN} \tag{B2}$$

where the coordinate transformation matrices T_{AI}, T_{IV}, and T_{VN} are defined in Appendix A. The vector, $\mathbf{g}^N$, is the acceleration due to gravity expressed in the N-Frame given by:

$$\mathbf{g}^N = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{B3}$$

where g is the earth's gravitational constant.

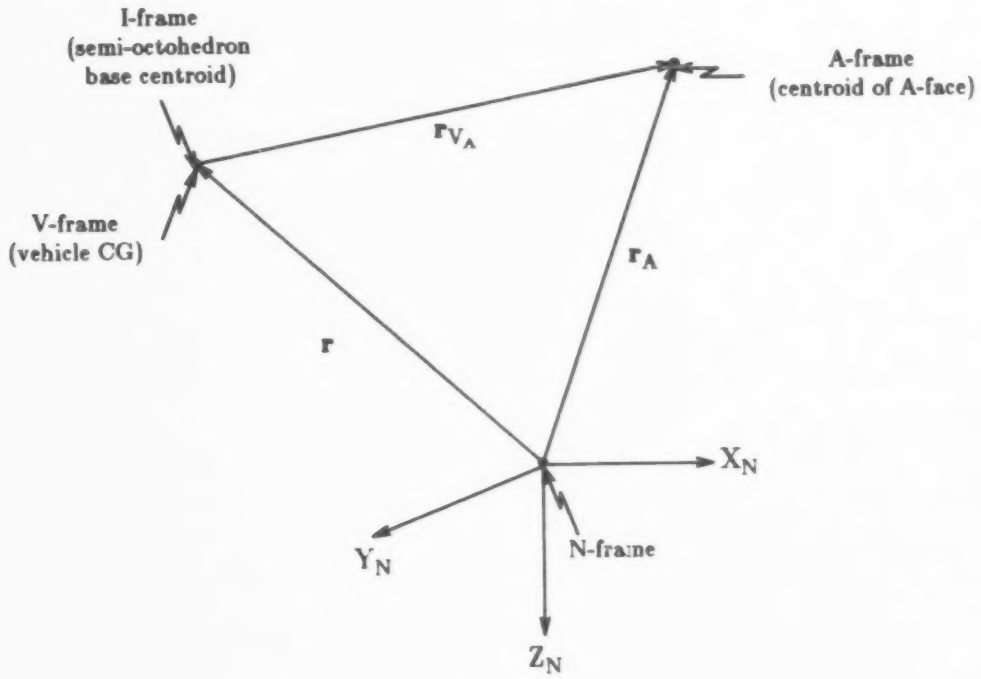


Figure B1. Relative Location of Reference Frames

6.2. Accelerometer Measurements

The two accelerometers on each triangular face of the semioctahedron are misaligned from that face by a small amount. For face A, transforming the specific force in the A-Sensor Frame given by (B1c) to the Measurement Frame of Reference on that face, we get:

$$\tilde{f}^A = T_{\tilde{A}A} f^A \tag{B4}$$

where $T_{\tilde{A}A}$ is the coordinate transformation matrix, mapping the A-Sensor Frame coor-

dinates to Measurement Frame coordinates on face A, defined in Appendix A. Denote the components of the specific force in the misaligned Measurement Frame on face A by:

$$\bar{\mathbf{f}}^{A} \equiv \begin{bmatrix} \bar{f}_{AX} \\ \bar{f}_{AY} \\ \bar{f}_{AZ} \end{bmatrix} \tag{B5}$$

The two accelerometer measurements on face A are given by the first and second components of the specific force vector expressed in the misaligned Measurement Frame. That is, the accelerometer whose input axis is aligned with the $\bar{\mathbf{x}}_A$ axis of the Measurement Frame on face A has the output $\bar{f}_{AX}$, and the accelerometer with input axis aligned with the $\bar{\mathbf{y}}_a$ axis of the same frame has the output $\bar{f}_{AY}$.

We have thus specified the true accelerometer measurements for face A. The specification for the accelerometer pairs on faces B, C, and D are similarly defined. The actual accelerometer measurements are scaled transformations of the true accelerometer measurements corrupted with additive bias and noise.

7. APPENDIX C - LEAST SQUARES VEHICLE ACCELERATION ESTIMATION

Here, we give a mathematical specification of the inertial vehicle acceleration least squares estimation problem which is to be solved by using various sensor subsets at various computational steps in the program. For illustration, consider the least squares estimation of the inertial vehicle acceleration from a redundant set of the eight accelerometers. In this section, unless otherwise noted, the computation frame in which the vectors are expressed is the Instrument Frame of Reference.

The raw accelerometer measurements are first converted into appropriate engineering units using the supplied scaled factor expressions. Next, the converted accelerometer measurements for each face are compensated for by misalignment, and expressed along the ideal Sensor Frame of Reference axes. The projections onto the Sensor Frame axis perpendicular to the triangular face are to be ignored. Hence, for instance, for face A, the compensated measurements would be given by using (A14b) in Appendix A and (B4-5) in Appendix B:

$$f_{AX} = \bar{f}_{AX} - \theta^{A}_{xz}\,\bar{f}_{AY} + \theta^{A}_{xy}\,\bar{f}_{AS} \tag{C1}$$

$$f_{AY} = \theta^{A}_{yz}\,\bar{f}_{AX} + \bar{f}_{AY} - \theta^{A}_{yx}\,\bar{f}_{AS} \tag{C2}$$

where $\bar{f}_{AS}$ is an estimate (not measured, to be supplied) of the specific force along the $\bar{z}_A$ axis of the Measurement Frame $\bar{A}$ of the linear accelerometer, and θ^{A}_{xz}, θ^{A}_{xy}, θ^{A}_{yz}, θ^{A}_{yx} are the small misalignment angles defined in Appendix A. Similar results hold for faces B, C, and D.

From the results in Appendices A and B, the compensated accelerometer measurements along the in-plane axes of the four Sensor Frames are related to the specific force represented in the Instrument Frame by:

$$\mathbf{y} \equiv \begin{bmatrix} f_{AX} \\ f_{AY} \\ f_{BX} \\ f_{BY} \\ f_{CX} \\ f_{CY} \\ f_{DX} \\ f_{DY} \end{bmatrix} = \begin{bmatrix} \mathbf{x}'_A \\ \mathbf{y}'_A \\ \mathbf{x}'_B \\ \mathbf{y}'_B \\ \mathbf{x}'_C \\ \mathbf{y}'_C \\ \mathbf{x}'_D \\ \mathbf{y}'_D \end{bmatrix} \mathbf{f}^{I} \equiv C\mathbf{f}^{I} \tag{C3}$$

where the superscript ' denotes a vector transpose and $\mathbf{y}$ is the measurement vector comprised of the compensated accelerometer outputs, the vectors — $\mathbf{x}_A$, $\mathbf{y}_A$, $\mathbf{x}_B$, $\mathbf{y}_B$, $\mathbf{x}_C$, $\mathbf{y}_C$, $\mathbf{x}_D$, $\mathbf{y}_D$ — are given by (A6a,b), (A7a,b), (A8a,b), (A9a,b), the vector $\mathbf{f}^{I}$ is the I-Frame representation of the specific force on the vehicle, and the 8x3 matrix C

represents the transformation mapping the specific force in the I-Frame to the compensated accelerometer measurements.

The equations above represent a set of eight linear equations in three unknowns (three components of the specific force vector **f** in the I-Frame). The least squares estimation problem is then to find **f**(1), **f**(2), and **f**(3) minimizing:

$$J = \sum_{i=1}^{8} (\mathbf{y}(i) - \sum_{j=1}^{3} C(i,j)\, \mathbf{f}(j))^2 \qquad \text{(C4)}$$

where the superscript on **f** has been dropped for notational clarity. You are to compute the solution, $\hat{\mathbf{f}}$, for the best estimate of the specific force in the instrument frame by solving the normal equations associated with this optimization problem. This solution is given by:

$$\hat{\mathbf{f}} = (C'C)^{-1}\, C'\ \mathbf{y} \qquad \text{(C5)}$$

where C' is the transpose of the matrix C, and the computation of $(C'C)^{-1}$ requires finding the inverse of the 3x3 matrix $C'C$.

Since the software is asked to compute the vehicle acceleration in the N-frame, the specific force estimate given above is transformed into the Navigation Frame and compensated for the gravity vector using the results in Appendix B to obtain:

$$\hat{\mathbf{a}} \equiv T_{NI}\hat{\mathbf{f}} + \mathbf{g}^N \qquad \text{(C6)}$$

where $\hat{\mathbf{a}}$ is the best estimate of the inertial vehicle acceleration $\mathbf{a}^N$, $\mathbf{g}^N$ is the gravity vector in the N-frame defined in Appendix B, and T_{NI} is the coordinate transformation matrix from the I-frame into the N-frame defined by:

$$T_{NI} = T_{NV}T_{VI} \qquad \text{(C7)}$$

where the coordinate transformations T_{NV} and T_{VI} are defined by (A4b) and in the paragraph following (A5b).

The preceding computations are to be performed by using the various sensor subsets as called for in the specification. For instance, if all of the four accelerometers on faces A and B are operational, then the acceleration estimate for Channel 1 is computed by performing the computations above with the 4x3 partition of the matrix C defined by the first four rows. Similarly, if the x-accelerometer on face B and y-accelerometer on face C are determined to be failed, then the best estimate of the vehicle acceleration is computed by performing the computations above with 6x3 partition of the matrix C obtained by deleting the third and sixth rows.

8. APPENDIX D - EDGE VECTOR TEST

Here, we give a mathematical description of the edge vector test, by specifying the edge vector relations to be tested, and the associated test thresholds. The edge vector test is based on the specific force measurements made in any two triangular face planes of the semioctahedron projected along the line of intersection of the two planes. If the two accelerometers on each of these two faces are functioning properly, then the two projections are approximately equal to each other within the constraints of the sensor noise characteristics. This edge vector test is formalized below.

In this appendix, all vectors are represented in the Instrument Frame unless otherwise noted. In addition, it is assumed that accelerometer measurements for a given face are expressed along the two in-plane axes of the Sensor Frame of Reference for that face, having been compensated for misalignments of the accelerometers. It is further assumed that the measurements have been converted to engineering units by the application of the appropriate calibrations to compensate for sensor bias and scale factor variations.

8.1. Edge Vector Relations

We denote the six unit length edge vectors by ($\mathbf{e}_{AB}$, $\mathbf{e}_{AC}$, $\mathbf{e}_{AD}$, $\mathbf{e}_{BC}$, $\mathbf{e}_{BD}$, $\mathbf{e}_{CD}$) where, for example, $\mathbf{e}_{AB}$ is the unit vector along the line of intersection of the planes containing faces A and B of the semioctahedron. We denote the accelerometer measurements in faces A, B, C, D by $\mathbf{f}_A$, $\mathbf{f}_B$, $\mathbf{f}_C$, and $\mathbf{f}_D$, respectively. There will then be six edge relations of the form:

$$\epsilon_{AB} = (\mathbf{f}_A \cdot \mathbf{e}_{AB}) - (\mathbf{f}_B \cdot \mathbf{e}_{AB}) \tag{D1}$$

where ϵ_{AB} is the evaluation of the edge vector relation for faces A and B, and the symbol $(\mathbf{a}\cdot\mathbf{b})$ denotes the scalar (inner) product of two arbitrary 3D vectors $\mathbf{a}$ and $\mathbf{b}$ and is given by:

$$(\mathbf{a} \cdot \mathbf{b}) = |\mathbf{a}|\ |\mathbf{b}|\ \cos(\theta_{ab}) \tag{D2}$$

where $|\mathbf{a}|$, $|\mathbf{b}|$ are the lengths of vectors $\mathbf{a}$ and $\mathbf{b}$ and θ_{ab} is the angle between the vectors $\mathbf{a}$ and $\mathbf{b}$. Hence, the first term in D1 represents the projection of the A-face specific force measurement, $\mathbf{f}_A$, onto the edge vector $\mathbf{e}_{AB}$, while the second term represents the projection of the B-face specific force measurement, $\mathbf{f}_B$, onto the same edge vector $\mathbf{e}_{AB}$. Similar expressions for the other edge vector relations $\mathbf{e}_{AC}$, $\mathbf{e}_{AD}$, $\mathbf{e}_{BC}$, $\mathbf{e}_{BD}$, and $\mathbf{e}_{CD}$. The components of the vector $\mathbf{f}_A$ along the A-Sensor Frame axes, $\mathbf{x}_A$ and $\mathbf{y}_A$, are given by (C1-2).

8.2. Test Thresholds

Due to the measurement errors in the accelerometers, the evaluation of the edge vector relations will not identically yield zero. You are to compare the evaluation of the edge vector relations with test thresholds defined by:

$$\delta_{AB} = \sigma_t \sqrt{(\mathbf{x}_A \cdot \mathbf{e}_{AB})^2 + (\mathbf{y}_A \cdot \mathbf{e}_{AB})^2 + (\mathbf{x}_B \cdot \mathbf{e}_{AB})^2 + (\mathbf{y}_B \cdot \mathbf{e}_{AB})^2} \quad \textbf{(D3)}$$

where $\mathbf{x}_A$, $\mathbf{y}_A$, $\mathbf{x}_B$, $\mathbf{y}_B$ are the A and B Sensor Frame axes defined in Appendix A and σ_t is defined by:

$$\sigma_t = \text{NSIGT x } \sigma_s \quad \text{(D4)}$$

where NSIGT is an integer specified from the set {3,4,5,6,7} and σ_s is the standard deviation of the accelerometer noise converted to engineering units.

8.3. Sensor Failure Detection And Isolation

Sensor failure detection and isolation is achieved by evaluating the status of the edge vector comparison tests:

$$|\epsilon|_{AB} \leq \delta_{AB} \quad \text{(D5)}$$

$$|\epsilon|_{AC} \leq \delta_{AC} \quad \text{(D6)}$$

$$|\epsilon|_{AD} \leq \delta_{AD} \quad \text{(D7)}$$

$$|\epsilon|_{BC} \leq \delta_{BC} \quad \text{(D8)}$$

$$|\epsilon|_{BD} \leq \delta_{BD} \quad \text{(D9)}$$

$$|\epsilon|_{CD} \leq \delta_{CD} \quad \text{(D10)}$$

For instance, if the second, third, and sixth relations above are satisfied while the first, fourth, and fifth relations are violated, then either the x-accelerometer or y-accelerometer on face B must have failed. The specific axis of failure is determined according to the procedure described in the specification.

Finally, note that the edge vector test applies to pairs of faces with all four accelerometers working properly prior to the invocation of the test. Once a specific accelerometer on a given face is determined to be failed, then all edge relations containing that face are to be discarded. For instance, if one of the accelerometers on face B is determined to be failed, then only the second, third, and sixth relations above are tested.

9. APPENDIX E - SOFTWARE & HARDWARE SUPPORT

This section describes the hardware and software which may be used in implementing the RSDIMU procedure.

9.1. Software

9.1.1. Operating System

The VAX 11/750 implementation of the Berkeley UNIX 4.2BSD system will be used for the development environment.

9.1.2. User Interface

For its flexibility, history facilities, and simpler shell script syntax, the C shell will be used.

9.1.3. Protection

Each team will have its own separate group 'universityid[A-F]' and software developed by that group *must* be stored under a directory with only 'universityid[A-F]' group read and write access. (The university identifier is the University's ARPA net or CS net address.) Unix group assignments will be announced when team assignments are made.

9.1.4. Use of File System

Software should be stored in a directory hierarchy, using the file system to support the software structure. Each directory should include a makefile for the software contained in that directory and a README file that documents the software structure represented by the directory. All code should be labeled using the dot convention: '.h' for header information, '.i' for include files, '.p' for Pascal, '.o' for object code, and '.t' for text processing source. Symbolic links may be used if required. Normal links should not be made to within the team directory.

9.1.5. Tools Set

9.1.5.1. Pascal

Berkeley Pascal (PC) will be used for program development. The ISO Pascal standard should be adhered to as a coding practice. No UNIX specific extensions may be used except for separate compilation; the separate compilation features of Berkeley Pascal can be used to simplify work.

9.1.5.2. Editor

Documentation will be provided for the screen-based text editor 'vi'. However, any available editor may be used.

9.1.5.3. SDB

Symbolic debugging of programs may be performed using SDB although there are some problems with this debugger. DBX only partially works for Pascal but may also be used.

9.1.5.4. Version Control

RCS will be used for version control because it provides fast retrieval of the current version. Every separate file storing a component of the software should be archived by RCS with a separate name and version number. Logging should be used and the log file kept up to date. Automatic version numbers should be maintained and these numbers should be included in the text of the program, as a character array constant in the object code produced for that text, and as text output of the program. The authors of the program should likewise include their name in the text, object code, and output.

9.1.5.5. Pascal Cross Reference and Pretty Printing

A Pascal Cross Reference option is provided by the program "pxp" and should be used to produce cross reference listings for the purposes of development. The program "pxp" can be used to remove include files and header files and produce a single Pascal program listing. The program may also be used to pretty print the Pascal.

9.1.5.6. Gprof: Profile program

Gprof may be used to obtain a run-time execution profile of a Pascal program.

9.1.5.7. Configuration Control

Makefile scripts should be used to support configuration control for ease of testing and compilation.

9.1.5.8. Mail

Most communications between principal investigator and students will be made by mail. Exceptions are discussions of hardware and software tools, and other subjects not directly related to the experiment.

9.1.5.9. Documentation Tools

Documentation techniques should be similar for every project. All text processing associated with documentation should be accomplished using the -me macros and nroff, troff, ditroff text processing systems. Tables should be prepared using 'tbl', equations written using 'eqn', and pictures drawn with 'pic'.

9.2. Hardware

9.2.1. Computing Machinery

All computing and document preparation will be done on VAX-11 series hardware.

9.2.2. Terminals

Your principal investigator will tell you which terminals are available for use.

9.2.3. Printers

Your principal investigator will tell you which printer(s) are available for use.

Appendix B. Student Question and Answer Sessions

1. Student Question and Answer Sessions

This appendix contains the questions submitted by students and answered by the coordinators during the conduct of this experiment, as well as the ten general announcements issued by the coordinators. These question and answer sessions were conducted over the electronic mail network. The absolute chronology of messages is not preserved here; the chronology within school is preserved by the numbering system, but not the chronology across schools.

1.1. North Carolina State University (NCSU)

ncsuq1) Page 16 of the specs, line 2: Isn't each accelerometer on face A for the whole A frame coordinate system?

ncsua1) The accelerometers are LINEAR accelerometers; each measures specific force along one axis. (See page 2 of specifications.)

ncsuq2) Same page, line 6: Where do they get the notation "X sub (I A bar)"? I don't understand why they have both I and A bar as a subscript, or mainly why the I is there at all, since they are talking about the misaligned axis and it seems that that should just be "X sub (A bar)".

ncsua2) First, the "line" over the subscripted A's here are not bars: they are (hard-to-read) tildes. To clarify the text and equations on pages 16 and 17, replace all occurrences of the subscript (I A tilde) with the subscript (A tilde).

ncsuq3) Page 58, line 7 (and other places): What exactly is meant by the term "engineering units"?

ncsua3) Units commonly used in engineering problems, like meters/second, degrees centigrade, degrees of arc, radians, etc. as opposed to units specific to some instrument, such as the "counts" used for accelerometers in this problem.

ncsuq4) page 61, The example of failed edge tests.
-- What do we do if a set of edge tests fail that have no face in common? For example, what if tests 1,4,6 failed on page 61?

ncsua4) If it is possible for this to happen, you would have to fail the system.

ncsuq5) page 26 Symbol to Segment table
--Several problems:
a) Segments for D are EXACTLY the same as the segments for 3.
b) There is no segment list for E.
c) In light of these problems I'm wondering if the segments for N are correct.

ncsua5) a) Segments for D should be BCDEG.
b) E is not a symbol to be displayed, therefore it needs no segment list.
c) Segments for describing the symbol N are correct as shown.

ncsuq6) page 3-4 How much should we question their doc set-up
-- On page 3 it says ".. input will include the calibration data set and a single set of sensor data values taken at a single time during the flight." Note that to do the calibration we need to calculate the SLOPE variable at the time of creation of the calibration data set. Will enough info be included in this set? The figure on page 4 leads me to believe otherwise since the variable TEMP is shown going into SCALE but not into CALIBRATE.

ncsua6) All global variables are available to your procedure upon its invocation. Feel free to create your own data flow diagram if you desire!

ncsuq7): The first eq. in p.16 of RSDIMU handout:
V(Xia) = (I(Xia)-2048)*409.6
I think the multiplication should be changed to division.
ie. => V(Xia) = (I(Xia)-2048)/409.6

ncsua7) You are correct. Change multiplication to division in the equation.

ncsuq8: From the definition of standard deviation, the denominator is n-1 instead of n. Should the denominator of S(Xia) of RSDIMU handout p.17 be changed to n-1?

ncsua8) No, the correct denominator is n. The equation with denominator of n-1 measures unbiased estimate of population standard deviation, and the equation with denominator n measures sample standard deviation. For the purposes of this problem, we intended to specify the denominator as n.

ncsuq9) Page 20 QUESTION: What does LINOUT hold?
It says that LINOUT holds acceleration components in the idealized Sensor frame. But page 34 states not to convert to this frame if there is only one functioning sensor on a face (leave it in the Measurement frame). So what does LINOUT hold?

ncsua9) Acceleration components in the idealized sensor frame if they are computable.

ncsuq10) Page 32 CLARIFICATION: calculating specific force after edge vec tst.
The bottom of page 31 states "compute a least squares estimate of the specific force on the RSDIMU ... using only those accelerometers on

faces determined to be good by the edge vector test." But later when calculating BESTEST we should use ALL functioning sensors, on good faces or bad faces. Is this correct?

ncsua10) Yes.

ncsuq11) Page 32 CLARIFICATION: SIGMA
SIGMA is referenced on page 32 but is not in the variable list.

ncsua11) References to SIGMA should be deleted. On page 32, please replace text from the sentence starting with "If the absolute value of this difference..." until the end of that paragraph with the following text:
"If the absolute value of this difference is greater than NSIGT x LINSTD for a given axis, then that accelerometer is declared to be failed. NSIGT is an integer from {3,4,5,6,7}."

ncsuq12) Page 58 QUESTION: Least squares and when to use equ C1 & C2:
Am I correct in assuming that, when sensors fail, it is possible for some of the sensor data input to the least squares solver to be in Idealized sensor coords and some to be in Measurement coords due to the ability to convert to idealized Sensor coords as stated on page 34?

ncsua12) Any single vector can be resolved into components in any coordinate system. The least squares solution assumes a uniform coordinate system.

ncsuq13) Page 32 CLARIFICATION: Sensor isolation and frames of reference
>From page 32: "(find) the projection of the computed specific force on these sensor axes (sensors on faces detected bad during edge vector test)." For these sensors I assume that "the projection" means to the Idealized Sensor frame. But for sensors on faces that had a bad accelerometer going into the edge vector test should this projection be to the idealized Sensor frame or the Measurement frame?

ncsua13) There is no basis for an edge vector test unless a face has two sensors believed to be functioning.

ncsuq14) Page 12 QUESTION: What is NORMFACE
What coord system is NORMFACE in, idealized Sensor or Measurement

frame? If it is in the idealized Sensor frame then shouldn't page 58 describe f(tilde, sub AS) as "not measured, calculated from supplied data" and not "not measured, supplied"?

ncsua14) Answered previously.

ncsuq15) Page 20 QUESTION: Algorithm structure and voters. The way I understand the function of the "fault detection and isolation" routine that we write it as follows. The goal is to calculate LINOUT, SYSSTATUS, and LINFAILOUT. Then we call the voter VOTELINFAIL, its what happens next that bothers me. I assume that before the voter call that LINOUT has non-zero values for ALL 8 sensors failed or not and LINFAILOUT indicates the failed sensors. The voter call is made, passing only SYSSTATUS and LINFAILOUT. LINFAILOUT may be changed, should we then zero the entries in LINOUT that LINFAILOUT (after voter call) indicates as failed? Or does LINOUT have zero entries before the voter call? If this is the case then if the call to the voter changes a failed sensor to functional status how are we to adjust LINOUT? Of course all this can be changed if the voter VOTELINFAIL accepts LINOUT as a parameter.

ncsua15) Answered previously.

ncsuq16) P 16, A silly typo; in the equation V = (I - 2048) * 400.6, * needs to be replaced by /.

ncsua16) Correct, replace with division.

ncsuq17) Are we to generate the offset values for sensors which are supposed to be failed even before the test begins?

ncsua17) NO.

ncsuq18) Above question repeated for the sensors which have been detected to be too noisy from the standard deviation.

ncsua18) Yes.

ncsuq19) The 3 words for displays are supposed to be Integers, are they going

to be treated as one or as bit patterns in consequent uses? If they are to be treated as Integers then the two most significant bits in first two and one bit in last word are undefined, and so will lead to four different yet correct answers for first two and two correct answers to last word. Is this acceptable?

ncsua19) The outputs are to be in word formats as given on page 27. Assume the display driver that would be a part of the final program extracts the necessary data from the word format and ignores the unneeded bits.

ncsuq20) In the 7-segment digit representation on page 25 should D be represented by ABCDEG instead of ABCDG and N by ABCEF instead of ABEF?

ncsua20) The correct representation for the symbol D uses segments BCDEG. The correct symbol for N uses segments ABCEF.

ncsuq21) What is the difference between type declarations (p. 37) SIARRAY and SMARRAY. There is no declared variable of type SIARRAY. In which context we are supposed to use it?

ncsua21) Remove the declaration of SIARRAY.

ncsuq22) Would there be two different sets of face temperatures; one for calibration and other for actual measurement? Or we are supposed to use the same value of slope in both the evaluations?

ncsua22) See NCSU Q28.

ncsuq23) In the 7 segment display, there is no provision for E. Can it be taken as ADEFG?

ncsua23) Yes.

ncsuq24) When displaying the hexadecimal representation of linear acceleration raw data, what is to be displayed in digit D2, as the data is only 12 bits long? Blank or Zero? or is it to be left justified instead? In which event what about D5?

ncsua24) Read the definition of the hexidecimal format and the count fields.

ncsuq25) While converting from one unit system to another (e.g. ft/sec2 to m/sec2) what level of precision is recommended?

ncsua25) Sufficient to preserve the accuracy of the data being converted.

ncsuq26) On page 34, if one accelerometer on a face has failed then the measurement from the other one (on the same face) should not be compensated for misalignment when used for channel estimates. Does this also apply to the BESTEST (estimate using all operational sensors)?

ncsua26) It cannot be compensated; it can be used.

ncsuq27) Page 29 There is a need for the digit "E" since modes 21-24 display the sensor input word in HEXIDECMAL!! So was your reply in error or should we not worry about modes 21-24?

ncsua27) See NCSU Q23.

ncsuq28) Page 15 Slope is temperature dependent, and TEMP is the current operating temperature of the face. Are we to assume that OFFRAW and RAWLIN are accelerometer counts taken at the same temperature or should there be two TEMP's? In other words is the face temperature taken on the ground while calibrating the same temperature during flight?

ncsua28) Yes.

ncsuq29) Page 54 Should we be concerned that T (sub A tilde A) times T (sub A A tilde) is not quite the Identity matrix?

ncsua29) No.

ncsuq30) Page 61 Is the conversion of LINSTD to what page 61 calls sigma (sub s) dependent on TEMP?

ncsua30) Yes.

ncsuq31) Q2: From the definition of standard deviation, the denominator is n-1 instead of n. Should the denominator of S(Xia) of RSDIMU handout p.17 be changed to n-1?

A) No, the correct denominator is n. The equation with denominator of n-1 measures unbiased estimate of population standard deviation, and the equation with denominator n measures sample standard deviation. For the purposes of this problem, we intended to specify the denominator as n.

****

Comment by mav: population standard deviation is given by sigma=sqrt(SSQ/N) where SSQ=sum((y-eta)**2) where eta is the true population mean (typically N is very large).

Sample standard deviation is given by s=sqrt(ssq/(n-1)) where now ssq=sum((y-ym)**2) and ym is sample average and n is the sample size.

If population mean is known then sample standard deviation is s'=sqr(SSQ/n).

Ref: e.g. Box et al. 1978, Statistics for experimenters, John Wiley, pp40-43 It appears to me that calibration values of the sensor responses do not fall into the class where population mean is known. It is only estimated through the average value. However if n i large, as I assume is the case for calibration data, then equation given on page 17 is a good approximation for the technically correct equation with the n-1 denominator (we have moved from small to large sample distributions).
Your answer gives impression that using n-1 would be incorrect while in fact equation on page 17 is an approximation of the unbiased estimate provided by the n-1 equation. Both equations (with n and with n-1) are dealing with sample standard deviations and are in this case estimating population standard deviation.

m.a. vouk

ncsua31) No answer required.

ncsuq32) Please refer to Eqn. D4. To convert LINSTD to Sigma sub s, we need to multiply by 1/409.6 and slope.
Question: What value of slope is appropriate? There are eight of them, one for each accelerometer.

ncsua32) A more detailed answer will be forthcoming.

ncsuq33)
Following is the copy of ncsu internal qa session (following your instructions that we should not send on already answered questions). However, doubt still dwells in minds of some of our students that the answer given below is correct. I need confirmation of that. Please let me know whether the answer given below is correct.
mav

>Internal (ncsuq31-2/12-June-85)
> In uiuca10 you stated that f tilde sub (AS) was the value
>supplied in the NORMFACE array. This implies that the NORMFACE
>values are in the Measurement Frame. In uiuca12 you state that
>the values are in the Sensor Frame. Which is correct?
>
>Internal(ncsua31-2/12-June-85)
>
> Values in NORMFACE array are in the sensor frame (see uiuca12)
mav

ncsua33) Transform the vector (oo a)' from the sensor to the measurement frames (or vice versa) and consider the value of the Z component of the result.

ncsuq34) In my understanding, the unit of RAWLIN(i) is 'counts'. If the unit of LINOUT(i) is 'counts' too?

ncsua34) The units of LINOUT are meters per (second squared).

ncsuq35) Could you please elaborate on uvaa2, I also misunderstood the phrase "previously failed sensors need not be considered in any computations."

ncsua35) Voter results will not contradict any given input variables.

ncsuq36) How does the above question apply to uiuc 24 which asks what causes LINOUT to be zeroed. When do we zero LINOUT, before or after a call to a voter? (please answer before of after)

ncsua36) After

ncsuq37) Since since the voters VOTELINOFFSET, and VOTELINFAIL seem to be able to overturn each others decision, what do we do about a zeroed entry in LINOUT that these voters declare as a functional sensor? (please answer 1) sensors declared as failed will never be undeclared in a call to a voter, 2) don't zero LINOUT until after the last applicable voter has been executed, 3) leave the entry in LINOUT zero even though the boolean sensor vector declares it operational, 4) other, please explain)

ncsua37) Voters will never contradict inputs. Voter returns may be considered as inputs for subsequent processing.

ncsuq38) I have not found where there is any mode for displaying the channel estimates in the output panel display section, modes 31-33 display BESTEST. Are we calculating CHANEST and never intend to display it or should there be a mode for displaying CHANEST? (please answer 1) you should calculate CHANEST but never display it, 2) use modes xx-xx to display CHANEST {please give format}, or 3) since you will never display CHANEST, you do not need to calculate it)

ncsua38) 1. The displays are not the only outputs of the program.

ncsuq39) In answer to uvaq22 you said to "Remove the words 'the average of' from lines 2 an 8 of page 32", by which I assume you mean lines 7 and 8. And this was in reference to the fact that LINSTD is a single number and thus cannot be averaged. While this is true there are more than one value of "LINSTD converted to program engineering units", one for each face. So should we keep page 32 the way it is or should we follow the answer to uvaq22?

ncsua39) In the context of P. 32, there is only one converted value of LINSTD for a given sensor.

ncsuq40) When a face is detected to have failed then we assume that an accelerometer on that face has failed. There now being a bad accelerometer on that face should we assume that the

conversion of the those accelerometer readings to the Sensor frame is in error and thus re-compute (from RAWLIN) the misaligned values for these sensors? Please note that if we use the "aligned" values then both sensors might be tested to have failed even though only one really has failed.

ncsua40) Alignment can only be performed with 2 good sensors. See UVACS A 42.

ncsuq41) While the answer to uvacsq 24 greatly simplifies the computation of delta on page 61, I need to know about the coordinate systems of x sub a, e sub ab, and x sub b. It seems to me in light of uvacsq 32 that the e vectors are in the I frame but the x vectors are in the Sensor frame, how can the dot product of these values be computed when they are in different coordinate systems?

ncsua41) See first sentence, paragraph 2, Page 60.

ncsuq42) In a related manner to the above question, can we compute the value of D1 in any manner we choose (ie convert the vectors to any convenient (sp?) coordinate system)? Or does D1 have to be compute with all vectors in the I-frame? Also, do we have to use D2 to compute the dot product, an easier method would be:

(a dot b) = (ax*bx + ay*by + az*bz)

ncsua42) The appendices supply background necessary to solve the problem. They do not specify coding details.

ncsuq43) Please refer to Eqn. D3.
(X sub A) dot (e sub AB) gives the component of (e sub AB) in the direction of (X sub A). Similarly for the second term. Now, (X sub A) and (Y sub B) are orthogonal. All three vectors are co-planar. So, the first two terms add to the square of (magnitude of e sub AB), which by definition, is one. The quantity under the radical sign reduces to 2. Question: Can D3 be re-written as
delta sub AB = sigma sub t root (2) ?

ncsua43) Previously answered UVACS 24.

ncsuq44) Please refer to (1)pg. 32, (2)Q & A session dt 9 Jun No. ncsuq11, (3)Q & A session dt 10 Jun No. uiucq8, (4)Q & A Session dt 10 Jun No. uvaq22. Do we or do we not delete references to SIGMA as specified in ref. (2), above?
If the change mentioned in ref. (2) is implemented (i.e., change Sigma to LINSTD), it should be cautioned that LINSTD is in raw units. If SIGMA is LINSTD converted to engineering units, the conversion will involve Slope. What value of Slope is appropriate? There are eight of them.

ncsua44) See NCSU 39.

ncsuq45) We are supposed to change LINSTD (standard deviation in counts) to SIGMA sub s (standard deviation in m/sec*sec) in order to dc the edge vector test (D4 p.61). To perform this conversion you need a slope (which comes from a temperature and 3 scale values) and an offset. We have 4 temperature values, 8 of each of the scale values, 8 slope values, and 8 offset values. How do we know which values to use to compute SIGMA, since none of those slope/offset values go with LINSTD?

ncsua45) To be answered.

ncsuq46) On p.29 we are told that when DMODE = 31, 32, or 33 we are to give the acceleration along the X sub N, Y sub N, or Z sub N axes respectively. Are we to use the acceleration computed in BESTEST, and if not, what values are we to use?

ncsua46) BESTEST

ncsuq47) Assuming that the MISALIGN angles are less than 0.1 radian (less than 5 degrees) and also assuming that an "average" force on the system is about 1-G (9.8 m/sec2) then let's let NORMFACE for this face be 5.20 and the x and y measured (not sensor) coords be 7.10 and 6.40, then if NCSU A33 says:

```
|  ?  |   | ¯ |  0  |   which converts <0 0 5.2>'
|  ?  | = | AA |  0  |   from sensor coords into
| z meas |   | 5.2 |   meas coords.
```

The obvious answer for z meas is 5.2 since the 3,3 entry of T sub a tilde a is 1 (see page 54, equ A13b)

My method which, for this example assumes theta sub ZY = 0.02 and theta sub ZX = 0.08 assumes NORMFACE is also in sensor coords but gives 4.83 for an answer a 3.7% error!!!

$$\begin{vmatrix} X \\ Y \\ N \end{vmatrix} = \begin{vmatrix} 1.00 & ????? & ????? \\ ????? & 1.00 & ????? \\ -0.02 & 0.08 & 1.00 \end{vmatrix} * \begin{vmatrix} 7.10 \\ 6.40 \\ ? \end{vmatrix}$$

sensor coords transform matrix meas coords

where N is the 5.2 above and ? is the misaligned z coord.
doing the last row of the matrix times the vector to get N we have

$5.2 = -.02*7.10 + 0.08*6.40 + 1.0*?$

solving for ? we have

$? = 4.83$

Now what do we do?

ncsua47) NORMFACE is in the sensor frames as it is NORMAL to the faces of the semioctahedron.

ncsuq48) Referring to NCSU A12 you said that "the least squares solution assumes a uniform coordinate system." It has already been established that when a sensor fails its partner is forever doomed to be in Measurement coords. On page 34 it states " if one accelerometer on a given face (has failed) while the other (has) not, then the functioning accelerometer on that face is used to compute channel estimates (and I assume BESTEST)..." therefore there will be a "non uniform" coord system going into the least squares solution, Measurement for the lonely sensor and Sensor for the pair in that channel that still work. Is this correct?

ncsua48) To be answered.

ncsuq49) Are we to worry about true transformations of various subranges? e.g. In the specs the references (for either face or sensor) is always made in terms of 1 to 4 or 1 to 8 etc. But since there are constants

declared for these purposes (SLB SUB etc.), can they assume any other values from their logical values? {They can be changed while testing the programs.}

ncsua49) Don't worry about this.

ncsuq50) For the display panel there is provision for values from 0.00001 to 99999. (same on negative side) what if a value out of these ranges is to be displayed? e.g. 100000? or it is impossible for such event? Also values in the open interval (-0.000005..0.000005) can be considered equal to .00000?

ncsua50) (Refers to the "Signed Decimal" format description on Page 28 as well as to Page 24.) If the value were out of range on the far negative end, -99999 would be displayed. If on positive end, +99999 would be displayed. Yes, values in open interval (-0.000005, 0.000005) may be rounded to .00000.

ncsuq51) p.28 (first sentence) refers to a special case of (blank, blank) for the Mode display. When is that applicable? What is the code for displaying that particular mode?

ncsua51) The first sentence on Page 28 is incorrect. Please delete the phrase "and a special case of (Blank, Blank)".

ncsuq52) On page 19 you say to report the accelerations in LINOUT as 'usual' when SYSSTATUS is false. What does 'as usual' mean, specifically, do we zero LINOUT according to LINFAILIN and LINNOISE, or do we report all values of LINOUT regardless of the status of LINFAILOUT, LINFAILIN, and LINNOISE, or do we zero LINOUT according to LINFAILOUT? Note that zeroing according to LINFAILOUT has two interpretations, the first is that all of LINOUT will be set to zero, and the second is that only some of LINOUT will be set to zero according to UVA 3. { Which says to set LINFAILOUT to true after voter call if SYSSTATUS is false }

ncsua52) When SYSSTATUS is FALSE, reporting the individual accelerations in LINOUT "as usual" means report them exactly as you would if SYSSTATUS had been TRUE. No changes to LINOUT are made solely based on the value of SYSSTATUS.

ncsuq53) In a related manner to the above question, since there will probably be 'valid' entries in LINOUT after all is said and done when SYSSTATUS shows up false, should we convert as much as possible of LINOUT to the aligned Sensor frame, or leave it all in the Measurement frame? { Note if the answer to the above question results in all of LINOUT being zeroed then this question requires no answer. }

ncsua53) Your question is answered in the description of LINOUT on Page 20.

ncsuq54) In Appendix E, we are cautioned to use 'no UNIX specific extensions.' Although the representation of the character set is not an extension, UNIX uses the USASCII character set. Does the procedure need to be portable or can we assume the UNIX 4.2BSD system and its character set?

ncsua54) Assume the UNIX 4.2BSD System and its character set.

ncsuq55) When displaying the RAWLIN values in hexadecimal, should we leave the sign blank or display a '+'? (since the value is in counts obviously not a '-')

ncsua55) Hexadecimal format does not use a plus or minus sign in this display; leave sign indicators blank.

ncsuq56) When using equation C5, can we assume matrices which will be found to have an inverse using standard numerical procedures such as Gaussian elimination with pivoting? If the determinant is found to be zero and cannot be compensated for, what should we do then?

ncsua56) The physics of the system should be taken into account in considering such a situation. This will lead to the conclusion that a singularity is quite unlikely. On the other hand, it should be made quite clear that any program which terminates abnormally for any set of "valid" input data is unacceptable.

ncsuq57) There seems to be considerable confusion as to what exactly LINOUT holds, which the seemingly contradictory responses to the various questions on the subject have done little to clarify. To illustrate: Refer to uiucq13 (10 Jun): " For all cases, should we use/pass the Unaligned value if only one sensor is working on a face ... This

applies to LINOUT". uiuca13: "Yes".

Now refer to uiucq40 (19 Jun):"... is Linout[1] supposed to be changed back to its original MISALIGNED value ...?". uiuca40: "No..."

Refer to ncsuq9 (14 Jun): To a question in the same vein, ncsua9 states "Acceleration components in the idealized sensor frame if they are computable". No mention is made of what is to be done if they are not "computable".

According to pg. 20 LINOUT should hold the acceleration in the "idealized Sensor Frame". However to calculate channel estimates, the misaligned values should be used for those sensors whose partners have failed (pg 3
Does LINOUT hold values only in the aligned (or Sensor) frame, or do we use some misaligned values as per pg. 34.

ncsua57) See ANNOUNCEMENT I.

ncsuq58) The answers given for NCSU 40 and UIUC 40 are apparently contradictory. Both questions ask what to do with the other accelerometer when one if found bad during the edge vector test. BEFORE GOING INTO THE EDGE VECTOR TEST IT WAS NOT KNOWN THAT THERE WAS A BAD ACCELEROMETER. Both accelerometers were aligned FOR the edge vector test. In NCSU 40 you said to change the other one back to unaligned coords while in UIUC 40 you said to leave it aligned. I have an example here that I admit is somewhat contrived but nonetheless shows what can happen if it is left aligned.

assume that LINSTD*NSIGT converted to engineering units is 0.1 m/s*s
the transformation matrix from unaligned to aligned is

1.00	-0.09	0.05
0.09	1.00	0.06
0.03	-0.04	1.00

and that the accelerometer readings (including the unaligned z) is
(7.31 6.25 9.30)'
while the true accelerations for that frame (unaligned) are
(6.25 6.25 9.30)'

obviously only the x accelerometer is bad, but if the converted values are used in the edge vector test both the x and y accelerometers will appear to be bad. Since

readings converted	7.21	actual converted	6.15
	7.47		7.37
	9.27		9.24

the difference between these is (1.06 0.10 0.03)'
which would indicate that both the x and y accelerometers are bad. This is something you have been assuring us will never happen.

Now, I ask you which answer do we use, the one to NCSU 40 or the one to UIUC 40?

ncsua58) Use ncsu40. Unfortunately, the answer to uiuc40 was incorrect.

ncusq59 Purpose of Question : to resolve conflicting mail on misalignment backtracking

Backgrounc of Dilemma :
To perform alignment, you must have 2 good sensors/face.

There are 4 conditions when a sensor can be declared bad or failed :

1. input (LINFAILIN)
2. noise (LINNOISE)

< ALIGNMENT >

3. if one sensor/face has been declared bad by 1 or 2 and then the other sensor on that face is found bad by least squares
4. (edge vector) detection & (least squares estimate) isolation

These tests are performed in the order specified by the numbering above. Alignment comes between 2 and 3. In case 3, the aligned values for the face with the failed sensor cannot be computed. In case 4, the detection and isolation of a failed sensor is performed after the alignment. A pair of sensors can pass the input and noise tests and be subsequently aligned and then one of the sensors can be failed during the failure detection and isolation routine. Alignment has been performed on the premise that both the sensors on the face were good.

The relation between alignment and sensor failure seems to imply backtracking. In his book, Software Engineering Concepts, (p.178) Richard E. Fairley defines "the need for lookahead" as "when processing of a data item depends on some characteristics of the yet-to-be-processed data."
Backtracking is the cure for lookahead problems. Alignment depends on go sensors. The functioning of sensors is not fully determined until after alignment. Therefore our procedure may be required to backtrack to the measured rather than aligned data.

There are the following 2 options for setting LINOUT elements :
(after voting on the results in LINFAILOUT)
if a sensor is now bad and was failed due to the failure detection and isolation algorithm then you can :

1. set failed sensor output to zero
   backtrack to misaligned value for functioning sensor of pair
2. set failed sensor output to zero
   retain aligned value for functioning sensor of pair

The following evidence is in support of backtracking (option 1) :

p.34 "If one accelerometer on a given face is determined to be failed" (doesn't specify when or how) ... "channel estimates are to be computed using the measurement not compensated for misalignment on the face with one functioning accelerometer."

uiccqa13 "For all cases, use/pass the unaligned value if only 1 sensor is working/face."

uvacsa42 "Misalignment compensation requires two good sensor values. Consider the effect of a bad value in equations A13 and A14."

ncsuq40 "Should we assume that the conversion of those accelerometer readings to the Sensor frame is in error and thus re-compute (from RAWLIN) the misaligned values for these sensors?"
ncsua40 "See uvacsa42."

The following evidence is in support of option 2 :

uiucq40 "Is LINOUT[1]" (represents the sensor which has not failed from context of question) supposed to be changed back to its original misaligned value for the purposes of output and for Vehicle State outputs?"

uiuca40 "No. As stated on Page 20, 'Values for failed sensors should be set to zero.' Also, see last paragraph on Page 59."

I apologize for the verbose format of this question however I wanted you to be aware of what I have read in the specifications and the mail and to have a more complete description of my understanding of the background surrounding the ongoing discussion of this question. If the following question can be answered without referring me to the above references and mail I would appreciate it.
I have been counseled by our experimenters to consider the mail chronologically, however uiuca40 seems to be in contradiction to a larger volume of previous mail and documentation.

******?QUESTIONS?******

Should we pursue option 1 or option 2 described above in this circumstance? If neither option applies as described, please elaborate on the faults of the description and endeavor to answer in the spirit of the question.

ncsua59) See Announcements I & III concerning LINOUT. Option 2 is wrong as the answer to uiucq40 was wrong (sorry). uiucq40 and this entire problem goes away due to the specification change in the Announcements.

ncsuq60) When performing least squares estimate test on faces with a sensor failed due to noise or input : If

1. All 4 sensors in pairs including that face are not good so edge vector test can't be performed
2. That face is not included in the "faces determined to be good by the edge vector test" (p.32) and so is not considered in the least squares estimate portion of the detection and isolation algorithm

Should the other axis on faces failed due to input or noise be tested before or after failure detection algorithm? (If either premise above is bad please specify which and why when answering, but also please specify before or after.)

ncsua60) Referring to your "1", see definition of SYSSTATUS for what happens when edge vector test can't be performed. Referring to "2", "before or after" is a design question, suffice it to say you are trying to detect if one accelerometer has failed (in addition to those failed to noise or input).

ncsuq61) 1) Is the gravity value of 32.0 ft/sec*sec correct enough to convert to m/sec*sec exactly as is? (Please answer yes or no.)
(In both this question and the next, please do not refer me to ncsua25. I am not asking how many significant places we should convert to, I am asking if the value 32.0 is correct, since we have found a value in a physics book of 32.2.)

ncsua61) Use the value provided in the specifications.

ncsuq62) 2) If the answer was no, how many decimal places should the value be taken to before conversion? Is the value considered to be measured at sea level? If not, what altitude should be used?

ncsua62) No answer required.

ncsuq63)/3_July_85

ANNOUNCEMENT II seems to be very simple, are you aware that this announcement and the answer to UVACS 24 imply that delta on page 61 of the specs is constant for all face pairs? Instead of averaging all eight values of SLOPE, why not use only those four values in question for the face?

ncsua63) The mathematical information in the Appendices is meant to be a guideline. You may use whatever method you choose that will obtain the correct answers.

ncsuq64)/3_July_85

Also, why average the slopes, I believe that the following method is superior since it reduces to the one given in simple cases and it also associates with each sensor the value of slope for that sensor.

delta sub ab =

sqrt[(sigma sub xa)**2 * (x sub a dot e sub ab)**2 +
(sigma sub ya)**2 * (y sub a dot e sub ab)**2 +
etc.

ncsua64) See ncsua64.

ncsuq65/3_July_85

I really hate to ask this question since it was probably just a typo, but in you answer to NCSU 50 referring to what to do if during the decimal display mode a value greater than 99999.0 were to be displayed you said to display 99999 my problem is the lack of a decimal point. Should it have been 99999. ? I realize that should the display ever read 99999 or 99999. that nobody would be alive to read it (10,000 + g's) but I still would like to know.

ncsua65) Yes, in ncsua50 replace "-99999" with "-99999." and replace "+99999" with "+99999." .

ncsuq66/3_July_85

I think you made an error in announcement 5 where you told us to add equations C8 and C9. The net result appears to be the following equation

y tilde sub B sup I =
T sub IB times T sub {{B tilde } B} times y sub B sup B

where y sub B sup B is [0,1,0]'.
As I understand the old method, we used the appropriate rows of the T sub BI matrix (or the columns of the T sub IB matrix) to convert a vector in the I frame to a reading in the Sensor frame. The intent of the new equation should be to use the appropriate rows of T sub {{B tilde} I} (or columns of T sub {I {B tilde}}) to convert a vector in the I frame to a reading in the Measurement frame.
The above multiplication does not appear to do that. In fact it is not clear what the above equation does at all.

As I understand the equation above, y in the Sensor frame is converted into the Measurement frame and then this vector in the Measurement frame is converted incorrectly to the Instrument frame. It is incorrect since T sub IB assumes that it is operating on a vector in the Sensor frame, when in fact it is operating on one in the Measurement frame. Shouldn't the result of these equations be

y sub {B tilde} sup I =
T sub IB times T sub {B {B tilde}} times
y sub {B tilde} sup {B tilde}

where
y sub {B tilde} sup {B tilde} is [0,1,0]'

that way we get the column of the matrix which converts a reading in the Measurement frame to one in the Instrument frame, or its transpose gives the row of a matrix that converts a vector in the Instrument frame into a reading in the Measurement frame.

ncsua66) No, these equations are used because in this case we can NOT compensate for misalignment while moving into the Instrument Frame of reference.

ncsuq67/3_July_85

in a related manner, using the old C matrix we had a nice form for C'C, namely it was symmetric positive definite. This

allowed a lot of very good algorithms to be used to compute the least square estimate. Now it is no longer guaranteed to be positive definite, and is harder to show that it is not singular. Will it be non singular or if we can't prove it should we test for it?

ncsua67) If you can't prove a quality you wish to make use of, you had better test for it. It is up to you to analyze the physics of the problem, and design accordingly.

ncsuq68/3_July_85

According to Announcement V, we will be computing estimates with misaligned and aligned measurements, which means that we will be doing the Least Squares Estimate on values in different coordinate systems. According to ncsu12 LSE assumes a uniform coordinate system. Is this problem taken care of by the fact that we are substituting new rows into the C matrix at the same time? If not, what is the solution to this dilemma?

ncsua68) As stated in the first paragraph of ANNOUNCEMENT V, the new method outlined there involves moving from the Measurement Frame to the Instrument Frame, so all will be uniformly expressed in the Instrument frame.

ncsuq69/3_July_85

Question uva61 seems to imply that we have to worry about the next cycle that the procedure will run through. Question ucla16 seems to imply the opposite. In either case, since the best estimate is done AFTER the edge vector test, the next time the edge vector test is done (which is supposedly what you are worried about saving BESTEST for) you will be figuring out whether you have enough good faces to do a edge vector test all over again. Why do you need to be concerned, when doing BESTEST, about the next run-through?

ncsua69) You are reading too much into uvacsq61. You may, if you wish, delete the phrase "i.e. that an operational face pair exists for performing an edge vector test in the next cycle", and the intent of the question as well as the answer remains the same.

ncsuq70/4_July_85
Can we assume that there will be exactly 50 previous values for each sensor in the array OFFRAW?

ncsua70) Yes.

ncsuq71) Sorry, I still don't like the answer to NCSU 66, let's consider it this way. When all sensors are functional the rows of the matrix C (on page 58) are taken from the appropriate rows of the matrices T sub AI, T sub BI, T sub CI, and T sub DI (see page 51).
(1) -- I assume that when there are some bad sensors there are some values in the Measurement frame and some of the rows of the C matrix should come from the appropriate rows of the matrices T sub {{A tilde} I}, T sub {{B tilde} I}, T sub {{C tilde} I}, and T sub {{D tilde} I}.

(2) -- T sub {{B tilde} I} is (for example)
T sub {{B tilde} B} times T sub BI
(3) -- It seems that ANNOUNCEMENT V is telling us to compute T sub IB times T sub {{B tilde} B}.

The answer to NCSU 66 is missing the point, it said that
"we can NOT compensate for misalignment while moving into the Instrument Frame of reference."
While closer examination of equation C3 indicates that we are NOT moving TO the Instrument frame, but FROM it.

This is not a casual method of looking at the problem, the rows of C are designed to convert f into the Sensor frame if all sensors are good, so it should convert f into the Measurement frame for some sensors whose mates are bad. C is not designed to move to the Instrument frame, but FROM it, never mind about the fact that f is unknown.

If the answer to NCSU 66 is correct, then one of the numbered assumptions above must be wrong. If so, please indicate which one is and why it is wrong.

ncsua71) The answer to this question, and ncsu66 if you like, is:
ANNOUNCEMENT V is an acceptable method.

ncsuq72) Consider the following hypothetical case.
Just prior to edge vector test invocation all the 8 sensors are deemed to be operational, thus making it possible to have all the six edge vector comparisons. If during this edge test if all but the last comparison fail, thus indicating face pair C and D to be operational, and A and B to be having a failed sensor on them. Is this scenario possible (as we are to assume that no more than ONE additional sensor failure will be detected at this stage)? If it is what is the system status, as the isolation of failed sensors on either faces is not possible?

ncsua72) See uiuca61.

ncsuq73) Since the square root part of Delta in D3 is just the square root of two and since announcement II said all the sigmas are the same why the big production on page 61 for something that is going to be a constant for all faces?

ncsua73) The Appendices contain mathematical descriptions which provide you with background information.

ncsuq74) : When doing an analytic solution for BESTEST you will apparently be converting 3 sensor readings into the Navigational frame. At that point do you add up all 3 x components to get the x component of the final answer (same with y and z) ? Or if not, what do you do? Also, does all this about the analytic solution apply to CHANEST too?

ncsua74) An analytic solution would be done the same way for CHANEST and BESTEST. By transforming (includes projections) the sensor readings to say, the I frame, you would have 3 equations in 3 unknowns, which you can solve for the x, y, and z components of the force...and take it from there as outlined in the specs, to compute the value of interest in the specified frame of reference.

ncsuq75/ 23 July
Is it possible that the test data answers given to us could be off in the z component of Bestest (BESTEST[3]) by the factor of gravity? In other words, that they forgot to add the gravity vector [0,0,G]' as an offset, as we are told to do in equation C6?

ncsua75) The test data answers were off in RAWLIN and OFFRAW by the factor of gravity. New test cases are forthcoming.

ncsuq76/ 23 July
I am confused by equation C6 on page 59 of the specs.
The following example illustrates my confusion, please indicate
where I have gone wrong.
1) If the vehicle is at rest on the ground then "a hat" in
equ C6 (BESTEST) should be the zero vector.
2) If the vehicle is at rest on the ground then T sub NI times
"f hat" in equ C6 should be <0 0 9.8>' since that is what the
sensors read. Note that positive Z is down.
3) Thus to compensate for gravity one should SUBTRACT <0 0 9.8>'
NOT add as shown in equ C6.

ncsua76)
You have gone wrong in 2) and 3). Gravity is exerting a force of [0,0,G]' upon the vehicle, since positive Z sub N is down. The sensors measure [0,0,-G]'. Think of it this way: since the vehicle isn't accelerating, in fact it isn't even moving, and a downward force of G is being exerted on it, it is as if the vehicle is accelerating upward, i.e. at -G. Thus to the sensors, the plane is accelerating at [0,0,-G]' in this case. Equation C6 is correct.

ncsuq77/ 23 July
In the test data's answers for BESTEST, we are given 6 digits on the right side of the decimal as accuracy. How many digits do we need to have exactly to pass the acceptance testing? If the voter uses another procedure such as averaging to determine the answer then could this answer be more accurate than it appears?

ncsua77) You will be provided with this information soon; but, in general, do not rely on any voters to increase the accuracy of the answer-- rely on the accuracy of your own answer only!!

ncsuq78/ 23 July
In UIUC 58 one of the assertions asked for confirmation was:
2) the values of the theta angles in the third row of matrices
A13b and A14b are zero.
the response was:
This is one way of looking at the problem, assertions confirmed.
In this case some values in MISALIGN are always zero, I can speak from experience that this is not so. Please comment.

ncsua78) Keep in mind that the combination of assertions was confirmed, and that it was ONE way of looking at the problem. Read uiucq58 carefully; it DOES NOT say that some values of the MISALIGN array will always be 0 on INPUT TO YOUR RSDIMU!

ncsuq79) After sending ncsua76, which showed everyone involved that equation C6 is correct, it seems to us after testing our program that the Creators of Test Data may have made a mistake. We believe that they did in fact do what they told us not to do in the above-mentioned question and have SUBTRACTED the gravity vector from (T sub NI * f hat) to get the BESTEST answers for test data set 3. Is this correct?

ncsua79) See uvacsa71

1.2. University of California at Los Angeles (UCLA)

uclaq1 On p. 61, the next to last paragraph, starting "For instance, . . . " we think there are other cases that need to be considered. For example, 1) D5 and D6 violated and all the others satisfied, and 2) D5 violated and all others satisfied. No 2 might happen when f sub A and f sub B deviate significantly and their deviations are additive. Therefore, only this one case fails (not the others involving f sub A and f sub B). Similarly no 1 above f sub A may be marginally bad.

uclaa1 See uvaa37, uvaa21.

uclaq2 re: uvacs42 and p. 34.
After detecting 1 bad accelerometer (through edge-test and least squares), do we use the other axis's measurement frame information in the rest of the calculations? For example, if we determine that the accelerometer on face A measuring the X axis is bad, should we use the measurement for the Y axis in face A from the measurement frame or the Sensor frame (using the adjustments for misalignment involving the BAD X axis accelerometer)?

uclaa2 Use measurements for the Y axis in face A from the measurement frame, in this case, after sensor failure detection isolation.

uclaq3 re: uiucq22.
We think the f in C4 is the f sup I in C3, in which case uiuca22 is wrong.

uclaa3 You are correct, uiuca22 is incorrect.

uclaq4 If LINFAILIN is TRUE, how should we set LINNOISE?

uclaa4 Don't do anything to LINNOISE for failed sensors.

uclaq5 I am not clear as what is meant by the average indicated value of the calibration, Ibar sub (X sub (A tilde)). Could you please clarify this?

uclaa5 PLEASE include page numbers and/or section numbers in questions!
Read section 2.3.3.3 (page 17) carefully, especially the sentence under the equation for S sub (X sub (A tilde)) and the definitions of OFFRAW and LINSTD. See also Pascal variables, page38.

uclaq6 Regarding the edge vector test on page 60, third paragraph.
The spec. reads , " We denote the six unit length edge vectors by (e sub AB, e sub AC, etc.) where for example, e sub AB is the unit vector along the line of intersection of faces A & B."
Do we have to calculate these unit vectors along the intersections of the faces OURSELVES?

uclaa6 The information in Appendix D should provide you with enough information to perform calculations needed for the edge vector test.

uclaq7 page 32,line 7. Could you give us a final version of all the changes made to this paragraph?!

uclaa7 See ANNOUNCEMENT II and uvaa22. (also ncsu13 for comment on 4th sentence in this paragraph.)

uclaq8 Is it true that when we find one face is bad by the edge vector test, we then run the same bad face through the least square estimate test to find out if the second sensor on the face in question was also bad or not?

uclaa8 Not quite, you don't understand what the least squares estimation does. Read CAREFULLY (don't confuse "face" with "axis") section 8.3, page 61, and last paragraph on page 31. See also uvaq17.

uclaq9 page 21. section 2.4.3.3 first sentence .
it says " five estimates of the vehicles state are produced"
what does this mean ?

uclaa9 Read definitions of CHANEST and BESTEST on page 22. See also Pascal declarations in section 3.

uclaq10 page 61 equation D4
does sigma (subscript s) has something to do with Linstd.
I don't know where to get the value for sigma(subcript s)

uclaa10 See ANNOUNCEMENT II.

uclaq11 page 17 fourth line
"I(sub x sub a) represents an individual element of this array "
is I array same as OFFRAW ?
what is the second dimension of OFFRAW(i,j) used for ?
what is meant by "element of this array " ? what array ?

uclaa11 OFFRAW(i,j) is the 8x50 array of calibration data. There are 8 sensors; there are 50 calibration points for each sensor.

uclaq12 page 16 second line
is g(sub x sub a) = 32 ft sec (square) -1 ?
does g(sub x sub a) mean gravational force on x axis of face A ?
is g(sub y sub a) = 32 ft sec(square) -1 also ?

uclaa12 Note the tilde over the A and see section 5.5, page 52. See also uvaa33, which makes it clear that g (sub X (sub A tilde)) is the projection of g along the X (sub A tilde) axis in the example. (Replace X's in above sentence with Y's for definition of same on Y axis.)

uclaq13 page 60 section 8.1
it says " e (sub ab) is the unit vector along the
line of intersection.."
how do we get this value e(sub ab) ?

uclaa13 The geometry of the problem is well defined. Study section 1.2 and the Appendices, especially equations D1 and D2.

uclaq14 page 58 equation c1
is f(bar)(sub as) = variable normface ?
if not , what is f(bar) (sub as) ?

uclaa14 Note the tilde: f(tilde)(sub AS) is an estimate of the specific force along the Z (tilde) (sub A) axis of the Measurement Frame of the sensor. Correspondingly, NORMFACE(1) is the supplied estimate, along the Z (sub A) axis, which is normal to face A of the semioctahedron.

uclaq15 page 32 second line
" estimate the specific force along the axis of each sensor in the questionable face by finding the projection of the computed specific force on these sensor axes".

How do we exactly " find the projection of the computed specific force " ?

uclaa15 If you don't know how to take a force represented in one coordinate system and project it onto another coordinate system, find a good introductory linear algebra textbook or teacher (but NOT a person involved in this experiment).

uclaq16 During each period of flight, one set of status and estimations will be generated. Is it right that LINFAILOUT will be the initial value of LINFAILIN of succeeding period?

uclaa16 Yes, that is the relationship of LINFAILOUT to LINFAILIN in the actual real time system. Since your program is not run in real time, this may not hold true for data sets your program runs with. See the last two paragraphs on page 3.

uclaq17 What meaning does the variable BESTEST.STATUS stand for? Is the following description correct?

- when more than 3 sensors are good, its value is NORMAL
- when exactly 3 sensors are good, its value is ANALYTIC
- when less than 3 sensors are good, its value is UNDEFINED

uclaa17 Taken out of context, yes, but see definition of SYSSTATUS, which by definition may influence BESTEST.

uclaq18 What is the difference between BESTEST.STATUS and SYSSTATUS?

uclaa18 Study the definitions of variables SYSSTATUS, BESTEST and the type SYSTEM. Their values convey much the same information, although expressed by very dissimilar Pascal types.

uclaq19 According to the specification in page 32, when only two sensors in two different faces failed, the edge vector test can not determine which of the two remaining faces fails when they do not agree. Therefore, the system would be declared nonoperational. I wonder that under this restriction of edge vector test, the system can only tolerate single-sensor failure and double-sensor failure in the same face. For a eight-sensor system, this seems to be not reasonable. Please clarify this point.

uclaa19 In a realtime implementation, sensor measurements are taken in such short time intervals that the probability of 2 sensors failing on the same sampling is very small. Also, the edge vector test is only one method for detecting failed sensors; other methods with different limitations exist.

uclaq20 For the display modes 31-33, the upper display will show the linear acceleration along the X, Y, and Z axes in the navigation frame. Do these values come from BESTEST?

uclaa20 Previously answered in ncsua46.

uclaq21 I did not see any display modes which would display the values of CHANEST. When the values of CHANEST are calculated, I did not see any procedures including display panel which would reference these values. Please clarify how the values of CHANEST would be used.

uclaa21 Previously answered in ncsua38.

uclaq22:On Edge Vector Testing page 61 section 8.3.
Assuming face D is bad prior to the edge test we discard D7,D9 ,and D10. Now if during the comparison testing D5, and D6 are satisfied while D8 is failed, what decision do we make ?
A computational error could have caused the above , however we cannot correct it since either face B, or C could be bad.
Do we set the system to nonoperational in this case ?

uclaa22 No, heed uvaa37.

uclaq23 please confirm the following statement:
Variable LINOUT is in the misalignment frame.
When we want to use LINOUT , we need to convert it to sensor frame.

uclaq23 Correct, the value you store in LINOUT is in the measurement frame.

uclaq24 When VOTELINFAIL says SYSSTATUS is FALSE, should the program terminate or try to do all it can (e.g. VOTEESTIMATES and VOTEDISPLAY)?

uclaa24 Your program should NEVER NEVER NEVER terminate just because SYSTATUS is false. You call all voters and assign values to ALL output global variables as outlined in the specs, no matter how many accelerometers have failed.

uclaq25 When doing number-crunching, particularly matrix inversion, there is a problem with introducing computational error. How sensitive are the voters to these errors? Should we be careful in our selection of number-crunching algorithms to reduce them?

uclaa25 You should take adequate measures to reduce computational error where you see a need for it. You may assume the voters do not Introduce any new computational errors. See uvaq37.

uclaq26 uclaa1 says see uvaa37 (which is not relevant) and uvaa21, which says see uvaa19, which says state the conditions stated in uclaq1.

uclaa26 Uvacsa37 is very relevant, and answers your unspoken question:
> uvacsq37 Consider the following example:
>
> During edge vector testing, relations D5 and D6 on p. 61 evaluate
> to FALSE, and relations D7 thru D10 evaluate to TRUE.
>
> According to the answer to a previous question, such an occurrence is
> MATHEMATICALLY impossible; however, the event does not seem to be
> COMPUTATIONALLY impossible. Real number manipulations on computers
> are subject to error, and error can cause theoretically impossible
> events to occur.
>

> In general, then, what do we do to avoid errors resulting from real
> number computations?
>
>
> uvacsa37 Use your judgment to convert the computationally possible answer
> to a mathematically possible one. In general, use sound numerical
> analysis techniques to identify and compensate for such situations.
> This is a program development exercise, not a coding assignment.

uclaq27 in ncsua 11 , the specification of NSIGT * Sigma is changed
to NSIGT * LINSTD.
in page 32 line 5. " if the absolute value of this difference is greater
than NSIGT * LISTD for a given axis, then that accelerometer is
declared to be failed".
The difference between actual and estimated specific force is m/sec square
but the unit for NSIGT * LINSTD is counts.
I think LINSTD should be converted to m/sec square as specified in
announcement 2 " to convert LINSTD to engineering unit, first convert
LINSTD to volts and then multiply this result by the average of the eight
slopes to obtain sigma (sub s)"
In other words, we should use NSIGT * SIGMA (sub s)
SIGMA (sub s) as in equation D4 on page 61, is that right?

uclaa27 Yes. You must pay attention to the chronological order of the
Q & A sessions; the ANNOUNCEMENTS superceded any conflicting
information that preceded the date of those ANNOUNCEMENTS.
Ncsua11 appeared before and conflicts with ANNOUNCEMENT II, and
therefore is invalid.

uclaq28 page 33 line 1-2
" outputs of this computation should be reported in the output variables
SYSSTATUS, LINFAILOUT, LINOUT"
from my understanding of the sensor detection and sensor isolation, we
don't output anything to linout. is this assumption correct ?

uclaa28 Due to the simplifications made in the ANNOUNCEMENTS, final values of
LINOUT may be determined before reaching the sensor detection
and isolation phase of the algorithm.

uclaq29 page 22 variables BESTEST and CHANEST
BESTEST is calculated in navigation frame of reference
and CHANEST is also calculated in navigation frame of reference.
right?

uclaa29 Right.

uclaq30 page 16 calculation of g sub x sub A tilde
I am confused about the unit of variable MISALIGN page 12.
To transfer from sensor frame to misalignment frame we
multiply the vector A (in sensor frame) by equation A13b page 54
to get vector A (misalignment frame).
Suppose vector A (in sensor frame) has unit meter/sec (square).
By multiplying equation A13b A(in misalignment frame) has
unit meter. radians/sec (square).
Are we suppose to ignore the radians ?

uclaa30 (A13b) is not an equation. I assume you mean (A14a). This equation
does exactly what the sentence above it says it does. Think about
the physics of the situation.

uclaq31 Referring to uclaa22, it says we shall not set the system to nonoperational
in that case. How shall we set the system and decide which face is good?
Please give us a more complete answer due question uclaq22.

uclaa31 Any numerical analysis techniques you use are up to you; you are
asking a design question, not a specification question.

uclaq32 As I saw in the ANNOUNCEMENT I & III, all LINOUT's which fail should
be set to be BADDAT. But at the bottom of page 19 in the specification,
it said that when SYSSTATUS is FALSE, all LINFAILOUT should be set to
TRUE, however, "The individual accelerations in LINOUT should be
reported as usual". This sentence seems to mean that we should not
change LINOUT's when system fails.

Please clarify the way we should treat LINOUT when system fails.

uclaa32 Announcements and Question/Answer sessions must be regarded in their
chronological order; they all, of course, make corrections to and
supercede the specifications. Announcement III makes it clear that
the only time an element of LINOUT is set to BADDAT is when the
sensor was failed on input to your RSDIMU.

uclaq33 Is the value of LINSTD relative to zero ? For example, for the given test data, value of LINSTD is 3. If we use the formula

Voltage = (IndicatedValue - 2048)/409.6

then LINSTD will have negative value. We think that in order to convert LINSTD into voltage, we should only divide LINSTD by 409.6.

uclaa33 You are correct.

uclaq34 We were told that our "results should agree to within 1 percent of the BESTESTs provided." BESTEST is a vector. Must each component be within 1 percent, or must the length and angle agree to within 1 percent, or what? It appears to me that if the real value is close to 0, the error percentage will tend to be large, although the absolute error is quiet small.

uclaa34 What was meant is that each of your BESTEST acceleration values should agree to within 0.01 of the corresponding BESTEST value provided in the data set.

uclaq35 Should we be concerned about the length of the variables ? That is, is it possible that our program is running well in our machine, but cause errors in other machines due to this problem?

uclaa35 This is possible. We would greatly appreciate your making variable names, etc. unique within the first 8 characters.

1.8. University of Illinois at Urbana-Champaign (UIUC)

uiucq1) Page 25: I believe the 7-segment LED driver for the letter D should be BCDEG rather than ABCDG. Please confirm it.

uiuca1) Confirmed. Letter D should be represented by segments ABCDG.

uiucq2) Page 51: the normal vector for x(sub. A sup. T) is identical to the one for y(sub. A sup. T). I have yet not had the time to do the analysis myself but from a first look it seems awfully suspicious.

uiuca2) (NOTE: The "sup.'s" {superscripts} are not T's, they are I's. The printer font is hard to read there.) Yes, a correction needs to be made. All equations on page 51 are correct EXCEPT (A6b), the equation for y(subscript A superscript I).
The matrix entries in the FIRST TWO ROWS need to be mutually exchanged here. The third row entry is correct as printed(-2).

uiucq3) The Z axis in the global coordinates (N) and vehicle coordinates (V) is down, whereas in the instrument coordinates it is up. Is this deliberate?
uiuca3) Yes.

uiucq4) Are we going to have plenty of data for each of the parts (modules) that we will write? If not so are we expected to generate data to test out our software?

uiuca4) You will be provided approximately four complete sets of input and expected output. You will receive this before the validation phase begins. Other than this data, you are responsible for your own test data.

uiucq5) Why is there no representation for "E" in the display table on page 25? (We assume ADEFG is proper.)

uiuca5) See NCSU Q23.

uiucq6) Is it possible to get an on-line copy of the requirements specification for perusal? It is easier to scan if one may use the built in pattern matching of vi as an indexing mechanism.

uiuca6) No. There is too much PIC, TBL & EQN in it.

uiucq7) Page 16; are "g(sub x (sub a tilde))" (line 2) and "g(sub x (sub ia tilde))" (line 6) supposed to be the same quantity, and if not what is the difference? (I think that line 2 "g(sub x (sub a tilde))" is a typo. Line 6 contains the proper symbol.)

uiuca7) Either notation can be used to refer to a face of the semi-octahedron.

uiucq8) Page 32; "SIGMA is the average of LINSTD converted to program engineering units,..." LINSTD is a simple variable so what is the meaning of this statement?

uiuca8) See UVA A22

uiucq9) The edge-vector test can only be applied to a face with two working sensors. We are also supposed to test faces with only one working sensor (by least squares procedure) to see if the one working sensor has failed. We were told to assume that only one sensor could fail due to the above tests. Once we find a failed sensor by edge-vector testing should we also do the other test?

uiuca9) Use your judgment.

uiucq10) Page 58; in equations (C1) and (C2) is "f tilde(sub AS)" the value supplied in the NORMFACE array?

uiuca10) Yes.

uiucq11) Page 61; the standard deviation of accelerometer noise is denoted by sigma(sub s). How is this quantity to be calculated and what exactly is it?

uiuca11) See Sections 2.3.3.3. and 2.5

uiucq12) On p12, it says that NORMFACE is the acceleration component normal to the face of the semioctahedron, i.e. the (idealized) sensor frame. However, on p58, it says that we're supplied 'f tilde sub {AS}' (EQN notation), which is the specific force along the 'z tilde sub A' axis of the measurement frame (of face A). The measurement and sensor frames are misaligned. Should NORMFACE be assumed to be along the measurement frame? Otherwise, it would mean that we would have to misalign this given value to the measurement frame in order to get 'f tilde sub {AX}' (of p58). It would make a lot more sense if NORMFACE is with respect to the measurement frame.

uiuca12) But it is not, p. 12 is correct.

uiucq13) For all cases, should we use/pass the UNaligned value if only 1 sensor is working on a face? (Otherwise, this would be corrupted.) This applies to LINOUT.

uiuca13) Yes.

uiucq14) Is 'sigma sub S' (p61) just 'S sub {X sub {I A tilde}}' (p17) converted to volts then to linear acceleration?

uiuca14) No.

uiucq15) Should we concerned (to a great extent) about execution efficiency? Storage efficiency? Or, should the emphasis be on structure and modularity?

uiuca15) You should be concerned with correctness.

uiucq16) On p17, a standard deviation formula is given. However, the correct formula (i.e. as defined in mathematics) has 'n-1' as the denominator rather than just 'n'. Is this intentional?

uiuca16) Yes.

uiucq17) In the specs manual, p14, Offset is the 2048 count mark that determines the zero of the accelerometer. Lower on the same page there is the definition of Linear Acceleration = Offset + Slope * Voltage; in pp18-19 it explains how to find the linear offset (expressed in meters per second square).

Please confirm the following:
The 2048 count offset has nothing to do with the linear offset. The 2048 offset remains constant always, whereas the Linear Offset is calculated during preflight. With this assertion things look right but I need to have it confirmed because the accelerometer offset comes into the calculation of the Linear Offset.

uiuca17) Right.

uiucq18) consider the following:
on pg. 28, middle... Hexadecimal displays the value of a 16 bit word (0000-FFFF)

contradicts pg. 28, modes 21-24:
"This is bits 11-0 of the sensor input word"
implying a 3 digit hex value (000-FFF). Since the type here is MINT, shouldn't page 28 say "bits 15-0"?

uiuca18) I assume you mean Page 29 for the modes. No. See Section 2.3.2.2.

uiucq19) What is the format of the input data? That is, could you give us an example input that includes calibration data, sensor data during the flight, and so on?

uiuca19) See Section 3.3.

uiucq20) Is the 'procedure' expected to read the input or may it assume that the input variables are properly initialized by an external program? This confusion is due to the fact that the requirements document refers to what we are supposed to design as a 'procedure' and not as a 'program'.

uiuca20) See Section 3.3.

uiucq21) From the eqns for misalignment compensation (eqns. C1 and C2 which are in reality eqnA14b with the z-axis information ignored) we get the 'correct projections' onto the Sensor Frame axis from the Measurement Frame axis. A careful examination shows that instead of the coefficient of f(tilde) sub(AX) being 1, it should be a factor close to 1 which depends on angles theta(sub XZ) and theta(sub XY) (see Fig A7). The way it is set up now the contribution of the unaligned f(tilde) sub(AX) to the f sub(AX) is the force itself. Is it assumed to be so because the coefficient is too close to 1, for simplicity, or what? Are we supposed to use the equations C1 and C2 as they are, or should we calculate the coefficient for a more correct answer?

uiuca21) The equations given are appropriate.

uiucq22) Manual, p.59 right under equation C4: 'where the superscript...' . Is subscript what was meant (like the subscripts AX, AY,...,DY in eqn. C3? If not, what is the superscript which has been dropped? There seems to be no reference to it.

uiuca22) Yes.

uiucq23) After looking through several varying answers regarding display representations, here is a summary of what I assume the "correct" representations to be:

Symbol	Segments
D	BCDEG
E	ADEFG
N	ABCEF

Are these or are these not the proper codes?

uiuca23) (?) Please include question numbers when you reference prior questions.

uiucq24) Does *any* type of sensor failure (input, excessive noise test, edge vector test) cause the corresponding entry in LINOUT to be set to zero?

uiuca24) Yes.

uiucq25) It's still ambiguous in which coordinate system NORMFACE is supplied. Is NORMFACE in the z-axis of the (misaligned) measurement frame?

uiuca25) There is only one axis which is always normal to a given face.

uiucq26) UVA13 implies the following. "The system can still be operational with only 2 available faces. The system is declared nonoperational only if the edge vector test (using the only pair of faces) gives a result indicating a failed sensor." Is this interpretation correct?

uiuca26) Yes.

uiucq27) Can we ignore UVA 2? This was addressed in NCSUA11 (6/9), where that passage was changed and clarified.

uiuca27) According to my records, NCSU11 and UVA 2 do not deal with the same subject matter. See below.

ncsuq11) Page 32 CLARIFICATION: SIGMA
SIGMA is referenced on page 32 but is not in the variable list.

ncsua11) References to SIGMA should be deleted. On page 32, please replace text from the sentence starting with "If the absolute value o this difference..." until the end of that paragraph with the following text:
"If the absolute value of this difference is greater than NSIGT x LINSTD for a given axis, then that accelerometer is declared to be failed. NSIGT is an integer from {3,4,5,6,7}."

uvaq2 Sensors marked as failed in LINFAILIN will be noted as failed in LINFAILOUT as well. The spec insists that we use the results of the vote on LINFAILOUT in the rest of our computations, but it also assures us that previously failed sensors need not be considered in any computations. How do we handle a voting return which marks as operational in LINFAILOUT a sensor which LINFAILIN has already flagged as dead. Is our input wrong if that happens or is the vote faulty?

uvaa2 You misunderstand the phrase "previously failed sensors need not be considered in any computations".

uiucq28) Related to the above, NCSSUA11 says to use LINSTD. Should this be converted into engineering units?

uiuca28) It should be used in a manner consistent with the computation required.

uiucq29) Since G is given as '32.0', this means that at most, all of our results can be accurate only to 3 digits. Is this intentional?

uiuca29) The conclusion of your first sentence does not follow from the hypothesis.

uiucq30) Do we have to calibrate failed sensors?

uiuca30) Not if they are given as failed in the input.

uiucq31) On p32, it says that detection of the failed sensor on a failed face uses "only those accelerometers on faces determined to be good by the edge vector test." Does this mean not to use the face that was just detected to be bad? Also, does it mean ALL the operational sensors, or only the operational ones on the good faces?

uiuca31) It means exactly what it says.

uiucq32) Is 32.0 supposed to be the "exact" value for G? If not, how many significant figures is it? It is essential to know this because we need to convert from feet/{second squared} to meters/{second squared} and that requires a conversion constant that we would have to provide for ourselves.

uiuca32) See ncsua25

uiucq33) Can we ignore UVAQ22? This was addressed in NCSUA11 (6/9), where that passage was changed and clarified.

uiuca33) No. UVAA22 supercedes NCSUA11; ignore NCSUA11.

uiucq34) NCSUA30 implies that 'sigma sub s' is LINSTD converted into engineering units. To this, the slope must be used. However, the slope is different for each sensor. Are we supposed to average the slopes?

uiuca34) See ANNOUNCEMENT II.

uiucq35) My question concerns what happens when you translate Normface[i] into the misaligned frame of reference for use in equations C1 and C2, p.58. In doing the translation, an X and Y component enters into f tilde(sub AS). What should we do with these new components? Should we

a) forget them and just use the new Z component in calculating f (sub AX) and f (sub AY)?

or

b) add the new X component to f tilde (sub AX) and the Y component to f tilde (sub AY)?

or

c) none of the above.

uiuca35) The transformation equation is given in A14a.

uiucq36) LINOUT is the only output variable on which we do not have to call a voter. Is this intentional?

uiuca36) Yes.

uiucq37) On p20, it says that LINOUT is "a real valued array representing linear acceleration component of sensor i (i=1 to 8) in the idealized Sensor Frame of Reference appropriate to the sensor." Does this mean that, for sensor 1 (on the misaligned X-axis of face A), LINOUT should be the X-component of the linear acceleration (as measured only by sensor 1) that has been transformed into the Sensor Frame?

uiuca37) Yes.

uiucq38) LINSTD is given in counts (p. 17). Is sigma(sub s), in eqsn. D4, p. 61 the same as LINSTD? If so, how is the value of LINSTD converted to engineering units? It can be converted to voltage by the eqsn on p. 14 but how is it then converted to m/s*s?

uiuca38) Hint: See ncsua30.

uiucq39) In the first paragraph of sec. 2.6 we are told to compute an "analytic solution" if we have an exactly defined system. Can this be done by using the least squares procedure from Appendix C in which the C matrix (eqsns. C3, C4, and C5) is 3 x 3? If not, how is this "analytic soln." to be computed?

uiuca39) No. The transformation of measurements from the nonorthogonal frame to the Navigation frame is the analytic solution.

uiucq40) LINOUT holds the linear acceleration component of sensors in the idealized Sensor Frame (p. 20). Suppose XA and YA (LINOUT[1] and LINOUT[2] resp.) were aligned. During the edge vector test it was found that the YA sensor was failed. According to p. 20 LINOUT[2] is now to be set to zero. Is LINOUT[1] supposed to be changed back to its original misaligned value for the purposes of output and for determining Vehicle State Outputs?

uiuca40) No. As stated on Page 20, "Values for failed sensors should be set to zero." Also, see last paragraph on Page 59.

uiucq41) Although NORMFACE is only the z-component, it's in the sensor frame. So, it has non-zero x and y components in the misaligned measurement frame. Should we take this into account when using the misaligned values?

uiuca41) Yes, if the Theta sub zx and Theta sub zy of the matrix in A14b are not equal to zero; see ncsuq47.

uiucq42) When a face has only one operating sensor, we're to use the misaligned value for that sensor. It's possible to "align" the value by assuming 0 for the failed sensor. (This would differ from the unaligned value by a small amount due to the z-component estimate.) Should we make this assumption (when appropriate) for calculating f sub AX (p58), etc?

uiuca42) See ncsu48.

uiucq43) We're not certain how LINSTD is converted to engineering units.
Since it's given as a standard deviation, my feeling is that
LINSTD (engineering units) = Slope x LINSTD (counts) / 409.6
(i.e. the offsets are left off). Is this correct?

uiuca43) See ANNOUNCEMENT II.

uiucq44)
The last two paragraphs of section 5.4 on pg. 52 specify the relationship between the I frame origin and the origins of sensor frames A, B, C and D. In the first paragraph of section 6.1, it is stated that "the lever arm effects due to the separation between the face centroids and the Instrument (I) Frame origin can be ignored." Does this mean that the information supplied by the global input variable OBASE is not needed for any computations in our procedure RSDIMU?

uiuca44) Yes.

uiucq45) We're not certain how LINSTD is converted to engineering units.
Since it's given as a standard deviation, my feeling is that
LINSTD (engineering units) = Slope x LINSTD (counts) / 409.6
(i.e. the offsets are left off). Is this correct?

uiuca45) See ANNOUNCEMENT II.

uiucq46) Should one output the raw data to the display even if that sensor is known to be failed (as when DMODE = 21..24)?

uiuca46) Yes.

uiucq47) Since Pascal does not have bit-wise operations can we use C Language subroutines to do the operations, and link them as externals? If this question has already been answered please point to the question(s) or the section(s).

uiuca47) NO!!!!!! See section 9.1.5.1. It is imperative these procedures be as portable as is possible; this is the reason for restricting your development environment and tools.

uiucq48) Why does misalignment compensation require two good sensors on a given face? Specifically, in (A13b), the transformation matrix has constant coefficients. Say that on face A the x sensor is good while the y sensor has failed, and that the x sensor yields an acceleration value a. This value, a, may be transferred to the sensor coordinate system for face A via the matrix product of the transformation matrix (A13b) with the vector <a, 0 , 0> transpose. None of this depends in any way upon the other sensor.

uiuca48) Misalignment compensation on a face requires two good sensors because it is the face, not just projections of force that are identified. Look at equation A14a and the estimates of force given by f sub x tilde, f sub y tilde, and f sub z tilde, and think about the physics of the situation.

uiucq49) On page 12, the units of MISALIGN are given in "milrads". Is this radians * 10 **-3 ? (y/n) If not, then what does this unit represent?

uiuca49) Yes, you are correct.

uiucq50) The design phase ends this week and we still do not have the analytical solution specs, the slope calculation specs, the revised sigma substitute calculation specs, and some more. Specifically, uiuc34, ncsu32, ncsu45, uiucq39, and uiucq42 have not been answered. Please provide us with answers to these questions as soon as possible as they pertain to important parts of the RSDIMU development.

uiuca50) See ANNOUNCEMENTS I AND II for answers to all but uiucq39 and uiucq42. uiucq39 and uiucq42 will be answered ASAP.

uiucq51) The answer in uiucq40 is -I think- incorrect. The question refers to LINOUT[1], not LINOUT[2]; therefore it should be set to the misaligned value. Please check your answer and answer again to this question.

uiuca51) See ANNOUNCEMENT I.

uiucq52) There have been many questions and answers with respect to the vote procedures (ncsuq15,35,36,37,52, uiucq36,37, etc). Mostly they concern the status of LINFAILOUT. In the following questions, please answer with a Yes or No following the question number:

(1) The voting routine does not change any of the contents of LINOUT;

(2) The contents of LINOUT if bad sensors are detected in the vector test are changed after the voting, according to the new values of LINFAILOUT but not SYSSTATUS;

(3) updating of LINOUT is done *only* after the voting;

(4) if SYSSTATUS is false after the voting, LINOUT is not changed, but LINFAILOUT is (all entries are set to 'true').

If any of the answers is 'No' please write a line or two to clarify why.

uiuca52) See Announcements I & III. According to the new definition of LINOUT, 1) True, 2) False, 3) False, 4) True

uiucq53) Please consider the following two questions:

>uiucq9) The edge-vector test can only be applied to a face with two working
> sensors. We are also supposed to test faces with only one working sensor
> (by least squares procedure) to see if the one working sensor has failed.
> We were told to assume that only one sensor could fail due to the above
> tests. Once we find a failed sensor by edge-vector testing should we
> also do the other test?
>uiuca9) Use your judgment.
>
>uvacsq40 p.32 "However, the health of the functioning axis of that face
> is determined according to the same procedure outlined above."
> Q. Is this procedure the one which is described at the top of
> page 32 beginning with the words "Using this computed
> specific force, estimate...?"
>uvacsa40 Yes.

Clearly, in uiucq9 the other test needs not to be done, since it would contradict the assertion that at most one *more* sensor would fail (otherwise single sensors in a face would be used in the test to find which axis has failed in a bad face but these sensors could be bad also-oops!).

With respect to uvacsq40 however, here are a few problems that could arise:
>From initial data (or failure of sensors during calibration) we may have only
one good sensor per face, with AT MOST one face having both sensors operational.
This means that there are no two faces known to be functional, which would
assure correct instrument frame acceleration calculations. In this case, if
we do the test of one sensor at a time using all other sensors for the
estimation, a bad sensor will be used in a test of a good sensor. The result
is that bad sensors will always come out as bad, but good sensors may come
out as bad also! There is a Catch-22 here. In fact, in the case of
exactly one bad sensor per face there is no way of telling whether one
sensor is failing the test because it is bad or because one of the sensors
used in the test is bad (in the case of one good face plus one sensor in each
of the other faces there is enough information to find what is going on
but the program would have to iterate through some tests --> too complicated!).
Is something like that within the scope of RSDIMU? Please answer in one
of the following ways: 1) If there are not at least two good faces the
sensor test should not be done for sensors not covered by the
edge detection test and they should be assumed to be correct, or,
2) sensors not covered by the edge detection test should always be checked for
functionality according to the following procedure (outline procedure
primarily in the case of no set of two good faces), or
3) (other) explain exactly what happens (e.g., from the data there will be

As a last remark, please let us know whether the letter 'F' should be used
for the display (same as edge detection test failures) or another one.

uiuca53) Congratulations, you are well on your way to solving the sensor
failure algorithm... however in a manner other than the method
we will use here. Your answers for choices 1) and 2) are false:
study conditions for the edge vector test and study the definition
of SYSSTATUS. Note that even if all fully functional faces used
in the edge vector test pass that test, there may be an as of
yet untested sensor on a face where the other sensor on that face
has been failed on input to your procedure. This is the case
uvacsq40 referred to.

uiucq54 According to ncsua47, NORMFACE is in the sensor frame. I believe that NORMFACE should be in the measurement frame.

Consider:

(1) Suppose that Tsm is a matrix that transforms a vector in the measurement frame into the sensor frame (say, A14b on p54) and Tms is a transformation matrix for the sensor to measurement frame (say, A13b on p54). Tsm and Tms should be inverses of each other. (A13b and A14b are not inverses because they are simplified due to the fact that misalignments are "small.") Now, given a vector Vs in the sensor frame, and a vector Vm in the measurement frame, the following is true:

(Tsm x Vm) + Vs = Tsm x (Vm + (Tms x Vs))

(2) The above situation is analogous to that of transforming the misaligned measured accelerations into the sensor frame while taking into account NORMFACE. If NORMFACE is in the sensor frame, then, one could just add the vector [0 0 NORMFACE]' (where the ' stands for transpose) to the resultant vector from the aligning of the measure accelerations. (Since the f tilde sub AX and f tilde sub AY of C1 and C2 on p58 supposedly take into consideration the misalignment components due to the transformation of NORMFACE into the measurement frame, as per uiuca41.)

This obviously means that NORMFACE contributes 0 to the X and Y components in the sensor frame. However C1 and C2 (the X and Y components in the sensor frame), have components contributed by NORMFACE (f tilde sub AS).

Thus, there is a conflict. So, how should we treat NORMFACE?

uiuca54 NORMFACE is as specified in ncsua47; in the Sensor Frame.

uiucq55 This question is remotely related to the above. The answer to uiucq42 was not quite adequate. In the case where only 1 sensor on a face is working, it is possible to assume a value of 0 for the other sensor, and then perform the transformation from the measurement frame to the sensor frame.

Assume that the X component sensor has failed. Looking at the transformation matrix (A13b on p54), it can be seen that assuming that the X component as 0 then transforming the Y component into the sensor frame and just using the value from the measurement frame would result in the same final value for the Y component value in the sensor frame ONLY IF both the X and Z components in the measurement frame were 0.

Now, if NORMFACE is in the sensor frame, its transformation into the measurement frame would introduce a component in the X-axis, in addition to having a value into Y-axis.

If NORMFACE is in the measurement frame, it would only have a value in the Z-axis.

However, using just the straight measured value from the Y component would different than transforming that value into the sensor frame assuming a 0 value for the X component.

So, we assume a 0 value in the failed sensor and convert the value measured by the good sensor into the sensor frame?

uiuca55 See ANNOUNCEMENT V.

uiucq56 uiuca39 states that the analytic solution cannot be done by using the least squares procedure. I beg to differ on this point.

The analytic solution is used when an exactly defined system of equations exist (p33); i.e., when exactly 3 faces are to be used.

When this condition occurs, the least squares method degenerates into just solving the system of equations (in this case, 3 equations and 3 unknowns) since:

(C'C) sup -1 times C' = C sub -1

if C is a square matrix (i.e. the number of rows and columns are the same).

uiuca56) uiuca39 is correct.

uiucq57) With respect to uiucq53, here is what I assume: the single sensor per face test is done only if both of the following conditions are met: The edge detection test found no bad sensor in any face (ie the at *most* one more bad sensor was not found), and, there are (up to this point) at least two good faces. Everything converges (at least in my mind) now: the single sensor test is done when enough information exists to make sure that we have a correct vehicle acceleration, otherwise SYSSTATUS becomes FALSE (by definition,p 19) in which case little do we care about the single sensors (the plane will crash anyway!).
This assumes also that ncsuq59 has the order wrong: the sequence is 1,2,4,3 with #3 being conditionally executed if SYSSTATUS is true *and* there were no failed sensors in #4.
The question is, do I have it straight?

uiuca57) This is one possible design; ncsuq59 outlined a different possible design.

uiucq58) Followup on ncsuq47: In my opinion, since from the specs and answers to pertaining questions (eg ncsua47) in the transformation matrices T (sub A-tilde A) and T (sub A A-tilde), ie A13b and A14b in p. 54, theta(sup A, sub zy) and theta(sup A, sub zx) are both zero; the reason is that the z axis is already in the sensor frame and needs no transformation. But the real interesting part is that IT DOES NOT MATTER: after we apply the transformation A14b to the measurement frame data we keep the x and the y components of the resulting vector as the x and the y components of the sensor frame data, and ignore the z aspect, which we do not need. Thus, the values of the two angles mentioned above are important only for the consistency of the model in the physical sense (ie there is something fishy otherwise).

This question aims to the confirmation of the following two assertions: 1) the measurement to sensor transformation depends only on the first two rows of matrix A14b, the first row yielding the x component and the second row yielding the y component when multiplied by the measurement frame vector (ie two misaligned values for x and y plus one from NORMFACE), and, 2) the values of the theta angles in the third row of matrices A13b and A14b are zero.

uiuca58) This is one way of looking at the problem; assertions confirmed.

uiucq59) There seems to be a consistency in the specs, namely, LINSTD is given in counts (p 17) and there is a count-to-voltage conversion factor (p14 l5), so there should be no problem on how to convert LINSTD to Volts, as required in Announcement II. However, LINSTD is supposed to be a standard deviation (as such it is supposed to be unitless from a mathematical standpoint). The question is, should we convert LINSTD to Volts by dividing it by 409.6 and not try to understand the model? If it is desirable for us to understand the model, please elaborate...

uiuca59) Yes, convert LINSTD to volts as described; see ANNOUNCEMENT II.

uiucq60)
Whereas the Announcements clarified what happens with the edge vector test they still leave unanswered what the threshold is for the single sensor isolation (manual, p32). Claim: the threshold NSIGT*LINSTD must be incorrect because even dimensionally it is not SI acceleration units. My best estimate is that the threshold MUST be the same as in the edge detection test, ie the average of the slopes multiplied by... (etc). After all, in both tests we want an algebraic sum of zero for accelerations, and if this is not zero there is only so much difference that we can accept. Please either confirm the assertion (that tolerance in the edge test is the same as in the single sensor test) or give us a specific formula for the latter case, which is dimensionally correct.

uiuca60) SIGMA on page 32 is the same as sigma sub s on page 61.

uiucq61)
Manual, p31, l15: 'Under the assumption that at most a single additional sensor will fail during execution...'. Does this imply that if the edge vector test succeeds in finding a failed sensor the single sensor test (as described in the previous question) does not have to be performed? Moreover, if it needs to be performed is it ensured from the data that at most one sensor will fail this test? This would be the specific case of two faces with one sensor each (due to input or noise), with the remaining two faces passing successfully the edge test.

uiuca61) You may make the assumption that "at most a single additional sensor will fail for a given execution", and design your program accordingly.

uiucq61)

Please confirm that in the calibration the sensors are aligned. If not, should we use the obvious transformation (A13b p54) for the misalignment, and for faces with one sensor only just use the direct value (no conversion)?

uiuca61)

Values in the variable OFFRAW are obtained from sensors on the semioctahedron, installed in the vehicle. See Section 2.3.3.2.

uiucq62)

The analytic solution is not entirely trivial, unlike what this note would lead us to believe:

>uiucq39) In the first paragraph of sec. 2.6 we are told to compute an "analytic
> solution" if we have an exactly defined system. Can this be done by
> using the least squares procedure from Appendix C in which the C matrix
> (eqsns. C3, C4, and C5) is 3 x 3? If not, how is this "analytic soln."
> to be computed?
>uiuca39) No. The transformation of measurements from the nonorthogonal
> frame to the Navigation frame is the analytic solution.

First of all mere transformation is not the correct process; transforming a vector essentially gives the same vector but with an x, y, and z rotation. Thus, the physical significance of the specific forces is lost if they are transformed only. Projections are needed. The projections to the Instrument Frame can then be transformed to the Navigation Frame. Projections directly to the Navigation Frame can be done by applying inverse transformations and then projecting, but this process is equivalent to the process I just described (projections to the I-Frame, then transformations).

There is a problem however, namely, that the Sensors (on the Sensor Frame) are not orthogonal to each other (with the exception of Sensors on the same face). Therefore orthogonalization has to be done PRIOR to the projection, or CORRECTION of the results (on the projection frame) has to be done, otherwise the results are total garbage.

Claim: if the least squares estimate is done properly (ie for an arbitrary partition of C, C-transpose and the y-vector) the eqn C5 in page 59 works properly for the analytic case, with the right half of the right side (product of C-transpose partition to the appropriate partition of the y-vector) being the non-corrected projection to the I-Frame and the left half of the right side (the inverse of the product of the C-transpose partition to the C partition) yielding the correction (voila'). Of course, in order to get the right answer we should use the generalized least squares estimation with such sensor status data that have *only* the three sensors in question in 'functional' status.

Question: In uiuca39 you specifically mention (1)Least squares estimate should not be used (even though as I described, it works), and, (2)Transformations but not projections are mentioned, which is -at best- incomplete (unless the math terminology is used loosely and it is implicit that transformations include projections). It is not clear what you want us to do. Please answer one of the following (with the appropriate explanation as needed):

(1) Ignore the uiuca39 and use the 'least squares' estimate of the three sensors in the analytic case, then transform the I-frame specific force to N-Frame specific force, and from that find the true force.

(2) Disregard uiuca39 and use any method you like (including the reduced to three sensors least squares estimate) provided that it yields the correct answer (specify if you need formal proof why it works).

(3) Even though the least squares estimate will work for three sensors it should not be used (explain why). The analytic solution should be computed in the following manner (explain exactly how, including orthogonalization method -eg Gram Schmidt- etc).

(4) The least squares method does not work (prove why). Explain *in detail* how the analytic estimation should be done.

uiuca62)

Use whatever method you know will work to calculate the analytic solution.

uiucq63 ncsuq63 and uvacsq64 are not questions about mathematical methods. They are specification questions. If one method uses the average of 8 slopes and another method averages only 4, the two will produce different results.

How do you want us to calculate the average slope (that is used to get sigma on p61)? Do you want us to use:

a) All 8 slopes in all cases,
b) Only the slopes of the working sensors (specify what is meant by "working" -- i.e. as indicated by LINFAILIN or LINFAILOUT),
c) Only the slopes of the sensors of the faces that make up the edge.

uiuca63 See ANNOUNCEMENT VII.

uiucq64 >uvacsa64 If you wish, delete the word "eight" from ANNOUNCEMENT II.
> Use your good judgment and heed ncsua63! Mathematical methods to
> use at some point become a design, not a specification question.

Related to the above, how do you want us to calculate the average slope (for getting SIGMA on p32)? Do you want us to use:

a) All 8 slopes,
b) Only the slope associated with the sensor being tested.

>uiuca60) SIGMA on page 32 is the same as sigma sub s on page 61.

uiuca64 uiuca60 is correct. Use the same method as given in ANNOUNCEMENT VII.

uiucq65 In the edge vector test, is it necessary for all the edge comparisons, that a face is involved in, to be violated before that face is detected as failed? For example, is it necessary for D5, D6, *and* D7 to all be violated before face A is declared as having a failed sensor (in the case where no sensors were detected as failed before this edge vector test)? (I.e. if only D5 and D6 are violated, face A is still considered as operational.)

uiuca65 It does not make sense that only D5 and D6 would be violated. If a sensor on face A is bad, it would show up in the D7 comparison as well as D5 and D6, as long as numerical computations have been done carefully. See uvacsa37.

uiucq66) This question is regarding the constant vector used to compensate for acceleration due to gravity.

"g sup N" of equation C6 on page 59 is defined in Appendix B (page 55, equation B3) as the transpose of [0, 0, G].
G is the acceleration due to earth's gravity.

I contend that g sup N should be

$$g^N = T_{NV} * g^V$$

where g sup V is the transpose of [0, 0, G]. The reason being that the accelerometers measure the specific force exerted on the vehicle, and consequently <0 0 G> vector can be used to compensate it only in the Vehicle Frame of Reference.

What is your response?

uiuca66 Wrong. (See paragraph 3 on page 2 for what accelerometers measure.)
Gravity exerts its force in the Z direction of the N Frame,
by definition. Therefore, g sup N = [0,0,g]'. See figures A2 and
A3a-b, and consider the effects of vehicle pitch and roll on gravity
as measured along the axes of the V frame.

1.4. University of Virginia (UVA)

uvaq1 Can we assume all input to our program is good, or should we build defenses against out-of-range input, input of wrong type, and the like? If so, what is the output for bad input?

uvaa1 Assume all global inputs to your program are good.

uvaq2 Sensors marked as failed in LINFAILIN will be noted as failed in LINFAILOUT as well. The spec insists that we use the results of the vote on LINFAILOUT in the rest of our computations, but it also assures us that previously failed sensors need not be considered in any computations. How do we handle a voting return which marks as operational in LINFAILOUT a sensor which LINFAILIN has already flagged as dead. Is our input wrong if that happens or is the vote faulty?

uvaa2 You misunderstand the phrase "previously failed sensors need not be considered in any computations". **********

uvaq3 If we compute SYSSTATUS to be false, do we set all elements of LINFAILOUT to true before or after we vote on SYSSTATUS & LINFAILOUT?

uvaa3 After

uvaq4 I had the same question about the representation of letter D on page 25. Your answer said confirmed and gave the representation as ABCDG. You did mean BCDEG, didn't you?

uvaa4 The correct representation of the display for the letter D is BCDEG. If my previous message on this subject was contradictory, I apologize.

uvaq5 What is a "skewed array" as mentioned in the first sentence of the second paragraph of page 2?

uvaa5 "Skewed" means 'set obliquely', or 'slanting'. There are eight accelerometers, to which we can refer as a "set of accelerometers" or, equivalently an "array of accelerometers". So we have a "skewed array of accelerometers" which is in this case a "set of eight accelerometers which are set obliquely on a semioctahedron".

uvaq6 For what is the mapping in the middle of page 11 (value -- face pair) used? What does this mapping tell us?

uvaa6 Change the word CHANEST on line 2 of page 34 to CHANFACE

uvaq7 the e eq. for voltage on p. 14 does not agree with what i think is the same eq. on p. 16. does voltage
= in (indic.val. - 2048)/409.6
or
= (indic.val. -2048)*409.6

uvaa7 The former

uvaq8 On p. 25, for SYMBOL "N" is the SEGMENT "ABEF" correct?

uvaa8 No, it should be BCEF

uvaq9 What is meant by 'skewed array' on p. 1?

uvaa9 See UVa Q 5

uvaq10 This is a minor point, but conceivably could be source of confusion.
The drawing on page 50 (Specifications) is slightly incorrect.
The marked position of the centroid is wrong.
The centroid of an equilateral triangle is located at the intersection of the angle bisectors, and that occurs approximately at the bottom vertex of the little square whose top vertex is marked as the centroid.

uvaa10 This is a schematic, not an engineering drawing

uvaq11 We don't understand why "the compensated measurement would be corrupted during the occurrence of a fault in the other axis" (p. 34). Since the transformation from the misaligned to the ideal sensor frame is independent of force measurements, why can't we do the transformation regardless of sensor status?

uvaa11 Your premise is wrong. See Eqn. A14a & UVa Q 15.

uvaq12 On pages 20 and 28 the text refers to "redundancy management software" without defining it. I take it that we write the RMS, and that it merely refers to the various BESTEST and CHANEST computations.

uvaa12 Sounds reasonable

uvaq13 Page 32, par. 3, sentence 2: Is the intended meaning equivalent to "Should the system be in a configuration where only a single pair of faces is available for the edge vector test (i.e., only two faces have both sensors operational), and the edge vectors subsequently fail the threshold test, the system must be declared nonoperational and no acceleration estimates can be made"?

uvaa13 Yes

uvaq14 Section 2.6, sentence 1: should "feature" be replaced by "failure"?

uvaa14 Yes

uvaq15 P. 34, par. 1, sentence 2: Is the intended meaning equivalent to "In these computations, misalignment compensation should not be performed because the compensation is invalid as a result of the failure of the other accelerometer"?

uvaa15 Yes

uvaq16 Does the reason for not performing misalignment compensation in the case of a failure of one accelerometer (p. 34) have to do with the lack of information regarding the Z-component of acceleration?

uvaa16 No

uvaq17 P. 31, par. 3: Must we guard against the possibility of more than one failure in a given execution?

uvaa17 No

uvaq18 P. 58, par. 2, sentence 2: Should the word "by" be deleted?

uvaa18 Yes

uvaq19 p. 61--the spec doesn't say what happens when some combination of d5-10 fails that doesn't unambiguously point to one and only one face as the culprit. what happens, for example, if only d5 and d6 fail? Should face a be declared faulty even though it passed d7? what happens if only d5 and d10 fail? what happens if d5, d6, d7, and d8 fail? etc. ad nauseum.

uvaa19 State the conditions under which this happens.

uvaq20 if sysstatus (p. 19) is defined to be faf true only when at least two faces are completely operational and their edge vector satisfied the threshold test, then at least four sensors must be working for the system to be operational. If four sensors are working the solution to bestest is normal; however, the specification of bestest (p. 33) considers the possibility of an analytic solution. how can this be? conceivably, three sensors on three different faces could fail due to noise, and the entire system would fail because it couldn't perform an edge vector test. does this make sense with five sensors operational as far as we know?

uvaa20 Yes.

uvaq21 is it mathematically assured that an edge the face which fails the edge vector test will contain a sensor that won't pass the test described on the top of p. 32? if not, what do we do if a face which fails the edge vector test contains two sensors which both appear to be working?

uvaa21 See UVa A 19

uvaq22 there seems to be an error in the description of sigma(p.32) it says that sigma is the average of linstd,but linstd is a single integer.how can you take the average of one number? should this be instead the average of the st.dev.'s calculated for each sensor that is used tc determine excessive noise during calibration ?

uvaa22 Remove the words "the average of" from lines 2 and 8 of page 32.

uvaq23 Is the 'offset' used near the top of page 14 the same as the 'offset' used near the bottom of page 14, and is it also the same as the 'offset' used in the equation in the middle of page 16?

uvaa23) No. The offset used near the top of page 14 shows the "standard offset" the accelerometers were manufactured to conform with for the purposes of representing voltages in "counts". In practice, this offset may not be exactly correct, due to the effects of temperature and other known forces. The offset used near the bottom of page 14 and in the the equation on page 16 refers to a correction to the "standard offset" which more accurately measures the actual current operating offset of the sensor.

uvacsq24 We are able to prove that (D3) page 61 of specs is reduced to

delta = (sigma_t) x (squared root 2)

in all the six cases (AB, AC, AD, BC, BD, CD).

What is the utility of (D3) as it is formulated ?

uvacsa24 No answer required.

uvacsq25 On p.16 of the specs, one reads in line 2

g sub X sub A bar

but on lines 6 and 11, one reads

g sub X sub I A bar

Why?

Also, along the same lines, what is the definition of

X sub I A bar

(or the X sub I A bar axis) in line 6? (As contrasted with the X sub A or X sub A bar axis, for example -- is some reference intended to expressing "misalignment" coordinates in the instrument coordinate system, or the like?)

uvacsa25 Answered previously. See UIUC Q7.

uvacsq26 Are we to assume that all data in global input variables is in the correct format, or do we need to perform data validation on the assumption that the 'instruments' supplying the data have malfunctioned?

uvacsa26 Answered previously. See UVA Q1.

uvacsq27 Where are the accelerometers located in relation to the Measurement Frame axes and the Sensor Frame axes? Are the accelerometers on the axes in one of these frames?

uvacsa27 See Section 1.2.

uvacsq28 Page 58, equations C1 and C2:
Is f tilde sub AS supplied by NORMFACE(i)?

uvacsa28 Answered previously. See UIUC Q12.

uvacsq29 Page 12: What coordinate system is NORMFACE(i) specified in?

uvacsa29 See Section 2.3.1.

uvacsq30 Where could we find information about the least squares estimation problem applied to sets of n linear equations in p unknowns, in order to check the veracy of (C5) page 59 of specs ?
We computed ((C'C)inverse) x C' , and the result seems a little too simple ...

uvacsa30 Try the library.

uvacsq31 On p.54 & other pages, various transformation matrices are given. (A14a) is supposed to transform a vector from the misaligned to aligned frame. This seems to mean that it finds the projection of the misaligned vector onto the aligned frame. Is this true ?

If this is true , then (A13b) should take the acceleration normal to the face ,

0
0
a (subscript Z)

and give its projection onto the misaligned frame. This however would return the projection of a(subscript Z) onto the misaligned Z axis as being equal in value to a(subscript Z). i.e the projection would have the same magnitude. This would mean that the acceleration in the Z direction was the same as in the misaligned Z direction.

Do I want to use the mathematical formula for using the projection,

projection = (T matrix)* ((T matrix transpose * T)inverse) *
(T matrix transpose)*(vector being transposed)

I believe that i want do to the latter, but I am not sure.

uvacsa31 The formulas A sub 13 and A sub 14 are appropriate.

uvacsq32 on p.60, it says to assume that vectors are in the i frame unless otherwise noted.at the bottom of the page it says that the components of vector f(sub a) along the sensor frame axes are given by (c1-2). should the vectors f(sub a) and f(sub b) in d1 be in the sensor frame or the instrument frame ?

uvacsa32 They should be in the Instrument Frame. Study equations A13 and C1 and 2.

uvacsq33 Should X sub A be X sub A-tilda on page 16 in the first complete sentence?

uvacsa33 Only if the preceding word "misaligned" is eliminated.

uvacsq34 On page 16, is g sub X sub A-tilda = 9.8 m/sec*sec? If not and we are supposed to use the equation on page 16 to calculate g sub X sub A-tilda, then how do we calculate Offset which is used in the equation for g?

uvacsa34 See the Pascal constant declarations.

uvacsq35 In view of answer ncsua11, should equations D3 and D4 on p.61 be changed?

uvacsa35 No.

uvacsq36: On page 16, how is g sub X sub A-tilda calculated given that we do not know the value of offset? or How do we calculate the value of g sub X sub A-tilda?

uvacsa36: Use the appropriate transformations of g. See Appendix 5.

uvacsq37 Consider the following example:

> During edge vector testing, relations D5 and D6 on p. 61 evaluate to FALSE, and relations D7 thru D10 evaluate to TRUE.

According to the answer to a previous question, such an occurrence is MATHEMATICALLY impossible; however, the event does not seem to be COMPUTATIONALLY impossible. Real number manipulations on computers are subject to error, and error can cause theoretically impossible events to occur.

In general, then, what do we do to avoid errors resulting from real number computations?

uvacsa37 Use your judgment to convert the computationally possible answer to a mathematically possible one. In general, use sound numerical analysis techniques to identify and compensate for such situations. This is a program development exercise, not a coding assignment.

uvacsq38 A question of convention:
On page 28 it is stated that in Hex format we are to display the value of a 16 bit word. Since only 12 bits of raw data are supplied, the most significant digit in this four digit Hex number will always be zero. Should this zero be displayed, or should a blank (nothing) be displayed?

uvacsa38 Display a digit 0-F in all four positions as required by the spec.

uvacsq39 p.32 "If a specific accelerometer on a given face is declared to be failed prior to the invocation of the program, then that face is not to be used in the edge vector test."
Q. Does this also include any accelerometers found to be failed due to excessive noise, not only those declared failed prior to the invocation of the program ?

uvacsa39 Yes.

uvacsq40 p.32 "However, the health of the functioning axis of that face is determined according to the same procedure outlined above."
Q. Is this procedure the one which is described at the top of page 32 beginning with the words "Using this computed specific force, estimate...?"

uvacsa40 Yes.

uvacsq41 If it is found that the functioning accelerometer on a face which was not used in the edge vector test is failed, may we assume that no other accelerometer will be found to have failed since "at most one more will be found to have failed after LINFAILIN and LINNOISE?"

uvacsa41 Yes.

uvacsq42 p.34 "..the measurement compensated for misalignment is not to be used since the compensated measurement would be corrupted during the occurence of a fault in the other axis."
Q.Throughout the discussion of the requirements, there are other calculations involving misalignment compensation seemingly without regard to the failure of the partner sensor. Is the problem mentioned on p.34 the only place where one sensor's being failed needs to be considered when performing calculations ?

uvacsa42 Misalignment compensation requires two good sensor values. Consider the effect of a bad value in equations A13 and A14.

uvacsq43 We are just trying to check out our understanding of the requirements. On page 58 we use equations C1 and C2 to do misalignment compensation before doing edge vector test ect. Do the results of C1 and C2 go into the output variable LINOUT ?

uvacsa43 As you have described using C1 and C2, the results could be placed in LINOUT. Please refer to Section 2.4.3.1., Page 20, and the paragraph above equations C1 and C2 on Page 58 for specific descriptions of LINOUT and the equations.

uvacsq44 Can you supply specifications for the Voter routines? The fact that their formal parameter names are the same as global variables (pp. 39, 40) is confusing.

uvacsa44 There is no reason for concern here. You do not need to know what happens inside the voter routines to design and code your procedure. All you need to know is that the voter procedure parameters are "var" parameters, what globals we want passed in what order, and where to place the calls to the voter routines. The specification, design and coding of voter routines are in the realm of the n-version test harness that all RSDIMU procedures will be executed in, not the realm of your team RSDIMU procedures.

(no uvacsq45 through uvacsq48)

uvacsq49 This question is in reference to uicuq9.
We understand that LINOUT holds acceleration components in the sensor frame if they are computable. We further understand that LINOUT(i) is set to 0 if sensor i is failed.
We would like to know what the value of LINOUT(i) is if sensor i is not failed but its partner sensor is failed thus making it impossible to compute the components in the sensor frame.

uvacsa49 See ANNOUNCEMENTS I and III for spec changes affecting LINOUT.

uvacsq50 It is stated on page 34 that "misalignment compensation cannot be performed [on a face with only] one functioning accelerometer."
Thus for those sensors whose values cannot be transformed from the measurement frame to the sensor frame, do we use the measurement frame values anywhere that we would have used the sensor frame values and just "pretend" that they are sensor frame values?

If not, then how do we get the values from the sensor whose partner has failed into the sensor frame so that they can be used in subsequent computations?

uvacsa50 See ANNOUNCEMENT V.

uvacsq51 Are the discussions of distance in appendix A directly relevant to the problem as it has been posed to us? Specifically, why do we need equations A1 and A12? Is OBASE necessary to the computations asked of our programs? If so, what for?

uvacsa51 The appendices contain background information that you may find useful. You are not obligated to use any specific equations as written if you find them unnecessary or know of other ways to obtain the desired results of the equations given. As for OBASE, see uiuca44.

uvacsq52 Concerning significant figures:

On page 35 of the specs, in the Pascal constant declarations, the value of G (gravity vector) is given as 32.0 ft/(sec sqr). Since this value must be converted to meter/(sec sqr) to be consistent with the rest of the data and computations, should we use the maximum precision of the computer in determining the converted value, or only use three (3) digits of the converted value (i.e., round to 9.75 meter/(sec sqr))?

uvacsa52 See ncsua25.

(No uvacsq53)

uvacsq54 Equation A12 defines the location of the origin of Sensor Frame A with respect to the I-Frame origin. Is the result of this equation used in any computation in our program (transformations, etc)? If so, where?

uvacsa54 See uvacsa51 and uiuca44.

uvacsq55 We were very surprised by the answer to uiucq37. Please confirm our interpretation of that answer. LINOUT DOES NOT contain acceleration data which has been compensated for misalignment. LINOUT DOES NOT contain acceleration data in the misaligned sensor frames. LINOUT DOES contain simple projections of the INDIVIDUAL sensor measurements on the X or Y axes of the sensor frames.

uvacsa55 See ANNOUNCEMENT I and III.

uvacsq56 One of the display modes requires knowledge of linear acceleration in the navigational reference frame. I assume that BESTEST is to be the source of that information. What happens if DMODE >= 31 and DMODE <= 33 and SYSSTATUS = FALSE? Should we leave DISUPPER and DISLOWER blank and display DMODE as usual? Please elaborate.

uvacsa56 BESTEST is used for modes 31-33. The value of BESTEST is fully specified; see definitions of SYSSTATUS, BESTEST, and text in section 2.4.3.3.

(no uvacsq57 through uvacsq60)

uvacsq61 CLARIFICATION:

The answer to uvaq20 seems to indicate that in calculating the status field of BESTEST, it is not sufficient to find any arbitrary set of four instrument values to declare status to be NORMAL. What is required is for the set of functioning accelerometers to include four which are working on some single pair of faces, i.e., that an operational face-pair exists for performing an edge-vector test in the next cycle. Is my interpretation of this question correct?

uvacsa61 Yes.

uvacsq62 If sensor i is failed at the time that LINOFFSET[i] is being calculated, what value does LINOFFSET[i] get?

uvacsa62 See ncsua17 and ncsua18.

uvacsq63 We understand the method of converting LINSTD to engineering units.
Part of this method requires us to take the "average of the eight slopes."
It does not make sense to use the slopes associated with
sensors which have been determined to have failed (either
from LINFAILIN or LINNOISE).
Should we average only those slopes associated with sensors
which have not yet failed?

uvacsa63 See ncsua63.

uvacsq64 We read the answer to ncsua63 before submitting this question.
It did not provide us with an appropriate answer to our
question. Therefore, we are submitting this question again
and do hope that you will answer it.

We understand the method of converting LINSTD to engineering units.
Part of this method requires us to take the "average of the
eight slopes."
Your clarification tells us to use all EIGHT slopes.
It does not make sense to use the slopes associated with
sensors which have been determined to have failed (either
from LINFAILIN or LINNOISE) even though you said to use all eight.
Therefore we would like to make sure that you meant what
you said.
Should we average all eight slopes or only those slopes
associated with sensors which have not yet failed?

uvacsa64 If you wish, delete the word "eight" from ANNOUNCEMENT II. Use
your good judgment and heed ncsua63! Mathematical methods to use
at some point become a design, not a specification question.

uvacsq65 It appears that the change in Announcement V directs us to
take a unit vector in a sensor frame, with sensor coordinates
(e.g., (0,1,0)) and first produce its components in the appropriate
measurement frame, then apply a change from sensor to instrument
coordinates to the result. Shouldn't we take a unit vector in a
measurement frame with measurement coordinates (e.g. (0,1,0) and
first produce its components in the appropriate sensor frame,
then apply a change from sensor to instrument coordinates to the
result.
We note that in ncsuq66 , essentially the same question is

asked, but the answer doesn't seem to resolve the difficulty. We are essentially asking if we should find components of unit vectors along measurement axes expressed in instrument coordinates, for cases where we can't perform the "alignment" of equations (C1)-(C2), p. 58. As they stand, equations (C9)-(C10) don't do this (because the first transformation is reversed --- the T sub {{ B tilde } B} of the example).

uvacsa65 See ANNOUNCEMENT VIII.

uvacsq66 In equations A13b,A14b,C1 and C2 are the misalignment angles provided to be used in milrads or they to be converted to radians ?

uvacsa66 Convert to radians.

uvacsq67 Is the value of BESTEST.acceleration equal to a-hat (as a-hat is defined in equation C6 on page 59)?

uvacsa67 Yes, as stated on the line following equation C6.

uvacsq68 Is the value of BESTEST.acceleration equal to f-N (as f-N is defined in equation B1b on page 55)?

uvacsa68 No. Read the line under equation B1c. Accelerometers measure specific force, but RSDIMU "estimates" are of inertial vehicle acceleration.

uvacsq69 Using your test data, starting with all sensors good, we changed the value of one element of RAWLIN slightly (only 18 counts), simulating a failure in a sensor big enough to fail 2 out of 3 edge vector tests but not big enough to fail all three.

Question:

Referring to uiuca65, when a sensor fails, will it fail by a large number of counts? In other words, what are the physics of a sensor failure? If the answer implies that either exactly 0 or exactly 3 edge vector tests will fail, then there is no problem. If the answer implies that we could get 1 or 2 failed edge vector tests as well as

0 or 3, is a failure of 1 or 2 edge vector tests to be considered to be a failure of a sensor? In other words, will a sensor always fail BADLY if it fails at all?

uvacsa69 In a realtime implementation, judgment would not be made on a single sensor reading; the time series of data would be considered, as not to fail a working sensor that at only one reading registered considerable noise. (Only that one reading would be ignored.)

Your procedures do not have this "inflight history" available. Therefore, they will be tested with data upon which failure decisions may be made from a single inflight reading; they will not be "too close to call."

uvacsq70 If sensors measure the effects of gravity, then the gravity offset required by C6 is unnecessary. If all raw data takes into account the effects of gravity, then the intermediate answers will contain the effects of this force. As a result, when the estimate is transformed to the N frame, it will carry with it the effects of gravity. Should we still compensate for gravity when the instruments have already done so? If the answer to this is YES, please explain why this apparent double compensation takes place.

uvacsa70 See ncsua76

uvacsq71 We agree that equation C6 is correct. However, we find that our answers to the test data differ from the supplied answers by approximately 2*G in the z-component. Is it possible that, contrary to equation C6, the supplied answers have been calculated by subtracting G from f sup N rather than by adding G?

uvacsa71 The problem was elsewhere, but there was a problem with gravity. You should have received new test cases (in files named "it<number>.dat"). You will have a day to run your procedure against these new test cases before submitting them for acceptance testing.

ANNOUNCEMENT I. Clarification on LINOUT controversy.

The following answers questions uiucq51, ncsuq57, and ncsuq59.

A SPEC CHANGE has been made to hopefully simplify and quell controversy surrounding what LINOUT holds and when it is calculated. This change supercedes all previously dated answers to questions on this subject. A new file of consts.h will be provided, containing the constant BADDAT.

Change the definition of LINOUT in section 2.4.3.1 "Linear Acceleration Outputs" to read as follows:

LINOUT(i) will hold real values representing the linear acceleration component of sensor i (i=1 to 8) in the MEASUREMENT FRAME OF REFERENCE appropriate to the sensor. Thus it holds raw acceleration values which have been converted to engineering units. Values for failed sensors should be set to BADDAT, a new constant defined as follows: BADDAT=9999; Units of LINOUT are meters per (second squared).

ANNOUNCEMENT II. Clarification on LINSTD controversy.

The following answers the questions uiuc34, uiucq43, uiucq45, ncsuq32, ncsuq45, uclaq10. This announcement supercedes any previously dated answers concerning LINSTD that are in conflict with this answer.

To convert LINSTD to Sigma sub s, you need to average the 8 slopes. To UPDATE YOUR SPECS, after the last sentence in section "8.2 Thresholds", add the following:

> "To convert LINSTD to engineering units, first convert LINSTD to volts and then multiply this result by the average of the eight slopes to obtain Sigma sub s."

ANNOUNCEMENT III. MORE ON LINOUT, UPDATE IN SPECS.

LINOUT will be computed as described in ANNOUNCEMENT I. Note that the only failures that will be reported in LINOUT are those provided as input in the variable LINFAILIN.

It has been decided that the value of the new constant BADDAT should be 9999.000. The values in LINOUT do NOT depend on any subsequent failures detected. This change affects the variable LINOUT only and not any other aspect of the specifications.

ANNOUNCEMENT IV. FRAMES OF REFERENCE USED IN LEAST SQUARES ESTIMATION

The following SPECIFICATION CHANGE supercedes any answers to questions posed in the past that conflict with this change.

There has been much concern over the possibility of some accelerometer measurements being in the Sensor Frame and others in the Measurement Frame of reference upon entry to the Least Squares estimation computation.

Least Squares estimation does assume a uniform coordinate system, so the statements on page 34 have been changed to accommodate the Least Squares Estimation procedure. Replace the full paragraph on page 34 with the following:

> "Note that if one accelerometer on a given face is determined to be failed while the other is not, then the functioning accelerometer on that face is used to compute channel estimates involving the face in question. In these computations, misalignment compensation is conducted in the normal manner as defined by equations C1 and C2, with the following exception: for the failed sensor, a value of zero is used. On subsequent entries into the program, the misalignment compensation is performed in this manner for the one functioning accelerometer on a face with one failed accelerometer."

ANNOUNCEMENT V. REVOCATION OF ANNOUNCEMENT IV!

The following SPECIFICATION CHANGE replaces ANNOUNCEMENT IV, since that method of compensating for the failed accelerometer is unacceptable in reality. Instead, a method for moving from the Measurement Frame to the Instrument Frame without compensating for misalignment is outlined, and is to be used.

Change, page 34. The full paragraph should read:

> "Note that if one accelerometer on a given face is determined to be failed while the other is not, then the functioning accelerometer on that face is used to compute channel estimates involving the face in question. In these computations, the measurement compensated for misalignment is not to be used, since the compensated measurement is invalid as a result of the failure of the other accelerometer. Hence, in this case, the channel estimates are to be computed using the measurement not compensated for misalignment on the face with one functioning accelerometer, as described in Appendix C."

Clarification, page 58. Equation (C3).

The eight vectors: x' sub A , y' sub A , ... x' sub D , y' sub D should all have a superscript I, to show that they are represented in the Instrument Frame. Add the (sup I) to each of the eight vectors in the explanatory text immediately below equation (C3) also.

Change, page 59. Append to the last sentence on the page:

"and by substituting y tilde sub B for y sub B, x tilde sub C for x sub C, and by using the uncompensated measurements f tilde sub BY and f tilde sub CX instead of f sub BY and f sub CX. The vectors y tilde sub B and x tilde sub C are the Measurement frame axes represented in the Instrument frame. and are to be determined from the results of Appendix A.

For example, to compute y tilde sub B we first compute y tilde sub B sup B via equation A13a as follows:

y tilde sub B sup B = T sub {{ B tilde } B} multiplied by y sub B sup B. (C8)

where y sub B sup B = [0,1,0]'. (The ' denotes a transpose.) We then compute y tilde sub B sup I via equation A11a as follows:

y tilde sub B sup I = T sub IB multiplied by y tilde sub B sup B (C9)

Similar computations hold for the other faces and axes."

ANNOUNCEMENT VI. INCLUDE FILES

When you submit your RSDIMU procedure for acceptance testing, it must NOT have ANY #includes in it. The files from the specification: consts.h types.h, vars.h and votes.h will be provided in the test harness. If you have any other #includes, please remove them and incorporate the necessary files into the file containing the procedure RSDIMU.

ANNOUNCEMENT VII. REWORDING PART OF ANNOUNCEMENT II FOR CLARIFICATION.

The following refers to announcement II, ncsuq63, and uvacsa64.

You may find by constructing realistic examples that it makes very little difference whether you average all eight slopes regardless of failures, or average some subset of the eight slopes, when calculating sigma sub s (page 61). However, to stop all controversy and insure a standard of engineering specification, please replace the last 6 lines in ANNOUNCEMENT II by the following:

To convert LINSTD to Sigma sub s, you need to average slopes. To update your specs, after the last sentence in section "8.2 Thresholds", add the following:

'To convert LINSTD to engineering units (Sigma sub s), first convert LINSTD to volts and then multiply this result by the average of the slopes of accelerometers determined to be good prior to the edge vector test.'

ANNOUNCEMENT VIII. REVISION OF ANNOUNCEMENT V.

The third section of Announcement V has been changed, due to inconsistencies noted in several questions from the Universities. Specifically, equation (C8) and its explanatory text are superceded by this Announcement.

REPLACE the section entitled:
"Change, page 59. Append to the last sentence on the page:" with the following:

"and by substituting y tilde sub B for y sub B, x tilde sub C for x sub C, and by using the uncompensated measurements f tilde sub BY and f tilde sub CX instead of f sub BY and f sub CX. The vectors y tilde sub B and x tilde sub C are the Measurement Frame axes represented in the Instrument Frame, and are to be determined from the results of Appendix A.

For example, to compute y tilde sub B we first compute y tilde sub B sup B via equation A14a as follows:

y tilde sub B sup B = T sub {B {B tilde}} multiplied by y tilde sub B sup {B tilde} (C8)

where y tilde sub B sup {B tilde} is the misaligned y measurement axis of the B face, expressed in the misaligned B Frame, and is given by [0,1,0]'. (The ' denotes a transpose.) We then compute y tilde sub B sup I via equation A11a as follows:

y tilde sub B sup I = T sub IB multiplied by
y tilde sub B sup B (C9)

Similar computations hold for the other faces and axes."

ANNOUNCEMENT IX. ANOTHER CHANGE TO ANNOUNCEMENT VIII

It has come to our attention that the information in Announcement VIII was not correct. In particular,

y tilde sub B sup {B tilde} = [0.1.0]'

is not true, since the {B tilde} axis is not orthogonal.

In fact, the equation (C3) is obtained by extracting the x and y components of the following equation (based on equation A10a):

f sup B = Tsub BI times f sup I

for the case with no failures on the B face, for example. For a face having a failure, the above equation is modified. For example, if a sensor on the B face is failed, we then have:

f sup {B tilde} = T sub {{B tilde} B} times T sub BI times f sup I

For a failure in the x axis of B face, that row of the C matrix is deleted and y' sub B sup I is replaced by the y (second) row of T sub {{B tilde} B} times T sub BI. Similarly, for a failure of a y axis on the C face, for example, that row of the C matrix is deleted and x' sub C sup I is replaced by the x (first) row of T sub {{C tilde} C} times T sub CI.

Therefore, the correction to the method to use is as follows:
Replace the change in Announcement VIII, from the paragraph beginning "For example, to compute y tilde sub B we first compute..." to the end of Announcement VIII with the following:

> "For example, to compute y tilde sub B we first compute the T sub {{B tilde} B} times T sub BI and take the second row of that matrix and transpose it into a column vector. Likewise, to compute x tilde sub c, we first compute T sub {{C tilde} C} times T sub CI, and take the first row of that matrix and transpose it to a column vector. Similar computations hold for the other faces and axes."

ANNOUNCEMENT X. CHECKLIST FOR PREPARING RSDIMU FOR ACCEPTANCE TESTING

The following is a list of things to check against your RSDIMU procedure before submitting it for acceptance testing. You will not pass the acceptance test unless these areas are properly treated.

1) CASE INSENSITIVITY. Your procedure must be written to be directly translatable into upper case. That is, by running your program through:
tr "[a-z]" "[A-Z]" < [your RSDIMU] > [output RSDIMU]

the resulting entirely upper case program [output RSDIMU] must run the same as the mixed case program.

2) ONE FILE ONLY. Your RSDIMU must be in one file only to fit in the test harness. NO #includes are allowed. Note that the global variables and voter routines will be provided in the test harness; DO NOT put them in your file containing the RSDIMU.

3) NO DEBUG STATEMENTS. There may be no read or write statements from your RSDIMU procedure. If you have debug statements in your procedure, comment out the debug statements prior to submission.

4) DOCUMENTATION. RSDIMUs should be well documented. At the very least we expect to see procedure headers. Additional inline commenting is welcomed.